ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

A Framework for Modelling Computational Sprinting with Phase Change Materials

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχίζας Β. Νικόλας

Επιβλέπων: Δημήτριος Σούντρης

Αναπ. Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

A Framework for Modelling Computational Sprinting with Phase Change Materials

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχίζας Β. Νικόλας

Επιβλέπων: Δημήτριος Σούντρης

Αναπ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Φεβρουαρίου 2017

.....	.....	.....
Δημήτριος Σούντρης	Κιαμάλ Ζ. Πεκμεστζή	Νεκτάριος Κοζύρης
Αναπ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2017

.....
Σχίζας Β. Νικόλας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικών Υπολογιστών Ε.Μ.Π.

Contents

Σύντομη Περίληψη

Ο όρος Dark Silicon προέρχεται από το γεγονός ότι, καθώς προχωράμε σε μικρότερες κλίμακες κατασκευής ημιαγωγών, όλο και μεγαλύτερο ποσοστό από την επιφάνεια των ολοκληρωμένων κυκλωμάτων θα παραμένει ανενεργό (dark). Το φαινόμενο αυτό προκύπτει από την αδυναμία προσαρμογής της ισχύος που απαιτείται για κάθε τρανζίστορ, αναλογικά με τη μείωση των διαστάσεων της συσκευής. Έτσι, σε τεχνολογικούς κόμβους της τάξης των micro-μέτρων, μπορούμε να κατασκευάσουμε περισσότερα ημιαγωγικά στοιχεία στον ίδιο χώρο χωρίς να υπάρχει αντίστοιχη μείωση της ισχύος που απαιτείται για τη λειτουργία καθενός από αυτά, με αποτέλεσμα, την αύξηση την πυκνότητας ισχύος στο κύκλωμα. Η αύξηση αυτή, μαζί με άλλους περιορισμούς που επιβάλλουν οι τεχνολογίες συσκευασίας και ψύξης των ολοκληρωμένων, εξαναγκάζει μέρος των κυκλωμάτων να μένει ανενεργό για να εξασφαλιστεί η λειτουργία εντός ορίων μέγιστης κατανάλωσης ισχύος (TDP). Η στροφή προς τις πολυπύρηνες πλατφόρμες ήταν το πρώτο βήμα σε μια προσπάθεια αύξησης της επίδοσης υπολογιστικών συστημάτων, κάνοντας χρήση του μεγαλύτερου αριθμού ημιαγωγών στο ολοκληρωμένο, η οποία οδηγεί σε μικρότερη αύξηση ισχύος από την κατασκευή ενός πιο ισχυρού πυρήνα. Ωστόσο, συστήματα με πολλούς πυρήνες χρειάζονται εργασίες που μπορούν να μοιραστούν ισόποσα στις διαθέσιμες επεξεργαστικές μονάδες για να επιτύχουν μέγιστη απόδοση. Επίσης, η αύξηση του αριθμού των πυρήνων εξακολουθεί να αυξάνει την κατανάλωση ισχύος και επομένως, δεν μπορεί να συνεχιστεί απεριόριστα. Συνεπώς, η έρευνα προσανατολίστηκε προς την ανακάλυψη νέων τρόπων για να χρησιμοποιηθούν τα ανενεργά κομμάτια υλικού (dark silicon) με στόχο την αύξηση των επιδόσεων. Μία από τις πιο υποσχόμενες ιδέες σε αυτή την κατεύθυνση είναι το computational sprinting. Η μέθοδος αυτή ενεργοποιεί πολλούς παραπάνω πυρήνες η/και επιταχύνει σε συχνότητα τους ήδη ενεργούς, για να επιτελέσει έντονες υπολογιστικά εργασίες σε χρόνο της τάξης υπό του δευτερολέπτου. Η διαδικασία αυτή ξεπερνάει για σύντομο χρονικό διάστημα τα προβλεπόμενα μέγιστα επίπεδα κατανάλωσης ισχύος και τερματίζεται όταν οι θερμοκρασίες στο ολοκληρωμένο φτάσουν σε κρίσιμες τιμές. Για να αυξηθεί ο διαθέσιμος χρόνος της μεθόδου, χρησιμοποιούνται υλικά αλλαγής φάσης (PCMs), σαν παθητική μέθοδος ψύξης. Η διπλωματική αυτή στοχεύει στον προσδιορισμό των βέλτιστων χαρακτηριστικών και της τοποθέτησης αυτών των υλικών. Για να επιτευχθεί αυτό, κατασκευάστηκε ένα μοντέλο προσομοίωσης υπολογιστικού συστήματος το οποίο αποτελείται από γνωστά εργαλεία, τροποποιημένα έτσι ώστε να μπορούν να προσεγγίσουν τα φαινόμενα που περιλαμβάνονται στη διερεύνηση με ακρίβεια. Τέλος, παρουσιάζεται μία νέα προσέγγιση όπου χρησιμοποιείται ένα στρώμα από υλικά αλλαγής φάσης με διαφορετικές θερμοκρασίες τήξης, με ενδιαφέροντα αποτελέσματα.

Λέξεις Κλειδιά : Dark Silicon, Computational Sprinting, Υλικά Αλλαγής Φάσης, Προσομοίωση Υπολογιστικού Συστήματος, Υλικά Διεπαφής, Ετερογενή Υλικά Αλλαγής Φάσης, Θερμική Προσομοίωση

Abstract

The term “Dark Silicon” derives from the fact that, as we advance to smaller technology nodes in the semiconductor industry, more and more chip area will remain unpowered (dark). This is a direct result of the failure to scale the switching power per transistor in accordance with device dimension shrinking. Thus, in deep, sub-micron technology nodes, we are able to pack more transistors at the same area without reducing the required power per transistor, resulting in growing power densities. This growth, coupled with the physical limits imposed by device packaging and cooling technologies, forces part of the chip to remain unpowered in order to operate within TDP limits. The shift to multicores was the first step taken towards gaining performance benefits from the bigger number of transistors on chip, that leads to smaller power consumption increases than creating a single, faster core. However, multicore scaling needs tasks that can be divided equally among cores to deliver maximum performance. Also, multicore scaling still results in growing power consumption and is therefore expected to come to an end. That being said, research is oriented towards discovering ways to leverage dark silicon area into performance gains. To this end, computational sprinting is one of the most promising ideas. Sprinting involves briefly powering on many extra cores and/or frequency boosting others, to facilitate sub-second bursts of intense computation. These bursts exceed power limits briefly and are terminated when temperatures on chip reach critical levels. In order to increase the available sprinting time, phase change materials (PCMs) are used as a passive cooling technique. This diploma thesis aims to determine optimal PCM characteristics and placement. To this end, a full system simulation model is constructed using well known simulation tools, augmented to be able to properly simulate the phenomena involved. Lastly, a new approach involving the use of a layer consisting of PCMs with different melting temperatures is presented with quite interesting results.

Keywords: Dark Silicon, Computational Sprinting, Phase Change Materials, Thermal Modelling, Full System Simulation, Thermal Interface Materials, Heterogeneous PCMs

.

Εκτεταμένη Περίληψη

Η διπλωματική αυτή εργασία έχει εμπνευστεί από ένα φαινόμενο που χαρακτηρίζει την τεχνολογία κατασκευής ολοκληρωμένων κυκλωμάτων το οποίο περιγράφεται με τον όρο «dark silicon». Το φαινόμενο αυτό προκύπτει ως αποτέλεσμα των διαδοχικών σμικρύνσεων των διαστάσεων των ημιαγωγικών στοιχείων (τρανζίστορ), τη στιγμή που, στις διαστάσεις πλέον των μικρομέτρων, δεν υπάρχει αντίστοιχη μείωση της ισχύος που καταναλώνει κάθε ένα από αυτά. Το φαινόμενο αυτό δεν είναι καινούριο και αποτέλεσε καθοριστικό παράγοντα στη στροφή της βιομηχανίας προς τις πολυπύρηνες πλατφόρμες. Οι πολυπύρηνες πλατφόρμες μας επιτρέπουν να συνεχίζουμε να αντλούμε κέρδη σε επιδόσεις χρησιμοποιώντας το περισσότερο υλικό που μπορούμε να χωρέσουμε στα νέα κυκλώματα, καταλήγοντας σε μικρότερη συνολική αύξηση ισχύος. Ωστόσο, οι πλατφόρμες με πολλούς πυρήνες χρειάζονται εργασίες που μπορούν να ισομοιράζονται μεταξύ των διαθέσιμων επεξεργαστικών μονάδων για να επιτύχουν τη μέγιστη απόδοση. Επιπλέον, ακόμα και η αύξηση του αριθμού των διαθέσιμων πυρήνων σε ένα σύστημα δε μπορεί να συνεχιστεί απεριόριστα. Έτσι, στους νέους, ακόμα μικρότερους κόμβους διαστάσεων, οι πυκνότητες ισχύων θα είναι πλέον τόσο μεγάλες που, για να διασφαλιστεί η λειτουργία του συστήματος σε επιτρεπτά θερμικά όρια, κάποια κυκλώματα θα αναγκάζονται να παραμένουν ανενεργά (dark). Συνεπώς, η έρευνα πλέον στρέφεται προς την προσπάθεια εύρεσης τρόπων να χρησιμοποιηθεί αποδοτικά αυτό το πλεόνασμα υλικού.

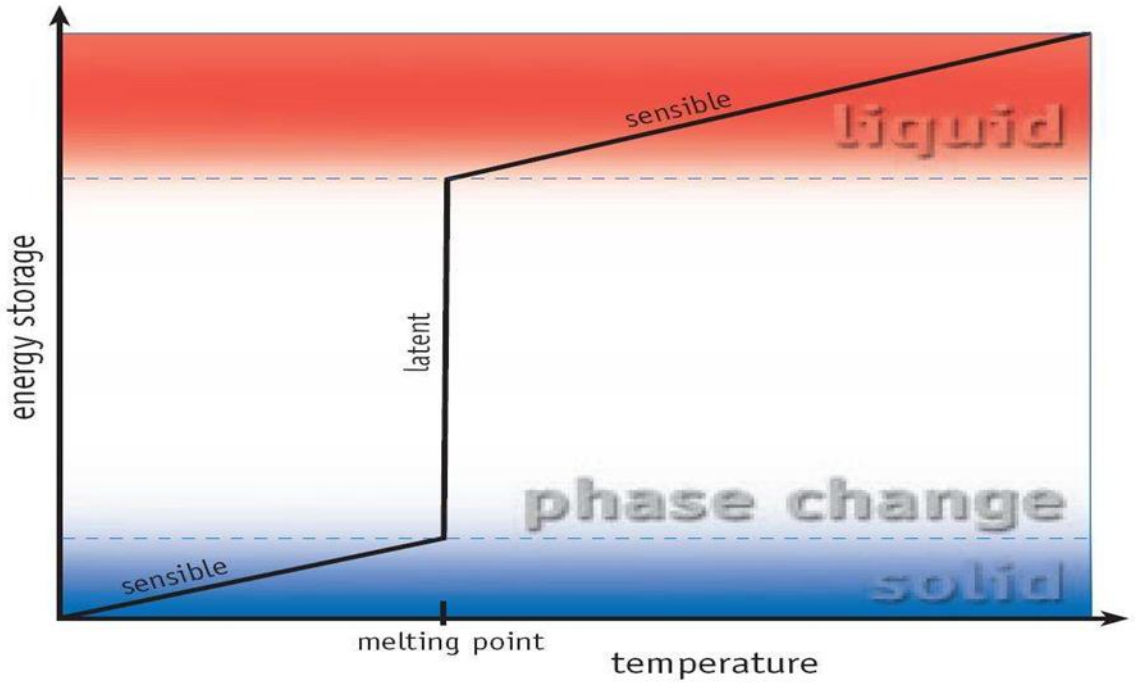
Από τις πιο διαδεδομένες τεχνικές προς αυτή την κατεύθυνση, είναι αυτή που αποκαλείται «computational sprinting». Η τεχνική αυτή, βασίζεται στην ιδέα ότι μπορούμε να ξεπεράσουμε για σύντομο χρονικό διάστημα, τα όρια μέγιστης κατανάλωσης ισχύος ενός συστήματος, χωρίς να προκαλέσουμε βλάβη. Αυτό, είναι άμεση συνέπεια της ενδογενούς θερμικής χωρητικότητας των υλικών ενός ολοκληρωμένου, δηλαδή, της ικανότητας των υλικών να αποθηκεύουν θερμότητα καθώς η θερμοκρασία τους αυξάνεται. Έτσι, όταν ένα σύστημα χρησιμοποιεί αυτή τη μέθοδο, είτε ενεργοποιούνται ανενεργοί επεξεργαστικοί πυρήνες είτε ενισχύονται σε συχνότητα οι ήδη ενεργοί ή και τα δύο. Όπως είναι αναμενόμενο, κατά την ενεργοποίηση των παραπάνω πόρων, ο ρυθμός παραγωγής θερμότητας από την πλατφόρμα ξεπερνάει κατά πολύ των ρυθμό με τον οποίο αυτή απάγεται προς το περιβάλλον. Το αποτέλεσμα είναι ότι υπάρχει ταυτόχρονη αύξηση υπολογιστικών επιδόσεων και παραγωγής θερμότητας. Ωστόσο, η θερμότητα αυτή αποθηκεύεται σταδιακά στα υλικά και η θερμοκρασία δεν αυξάνεται ακαριαία. Έτσι, υπάρχει ενα χρονικό παράθυρο στο οποίο μπορούμε να αυξήσουμε δραματικά τις επιδόσεις του συστήματος και μετά να επιστρέψουμε στην κανονική λειτουργία. Προφανώς, το τέλος του χρονικού αυτού παραθύρου σηματοδοτείται από την αύξηση της θερμοκρασίας σε κρίσιμες τιμές. Μετά από αυτό το γεγονός, οι παραπάνω επεξεργαστικοί πυρήνες απενεργοποιούνται και το σύστημα αφήνεται να αποβάλλει σταδιακά στο περιβάλλον την πλεονάζουσα θερμική ενέργεια και να επιστρέψει σε ονομαστικές θερμοκρασίες λειτουργίας.

Σε συνεργασία με το «computational sprinting», χρησιμοποιούνται και τα λεγόμενα υλικά αλλαγής φάσης. Στη γενική περίπτωση, τα υλικά αυτά είναι ενώσεις που έχουν την ικανότητα να αποθηκεύουν μεγάλες ποσότητες θερμικής ενέργειας κατά τη διαδικασία αλλαγής φάσης. Στη

σύνηθη περίπτωση, εκμεταλλευόμαστε την αλλαγή φάσης από στερεό σε υγρό και αντίστροφα. Από αυτή τη σκοπιά, τα υλικά αυτά είναι ουσίες με υψηλή θερμότητα τήξης. Η θερμότητα τήξης ή λανθάνουσα θερμότητα τήξης, είναι η θερμική ενέργεια που απαιτείται για να μετατραπεί μια συγκεκριμένη μάζα ενός υλικού από στερεό σε υγρό. Ο όρος λανθάνουσα, υποδηλώνει το γεγονός ότι η αλλαγή φάσης πραγματώνεται σε προσεγγιστικά σταθερή θερμοκρασία, το μέτρο της εσωτερικής θερμικής ενέργειας μίας ουσίας, επομένως, το υλικό απορροφά ενέργεια η οποία λανθάνει εξωτερικής παρατήρησης. Θερμότητα η οποία απορροφάται από ένα υλικό και προκαλεί παρατηρήσιμη θερμοκρασιακή διαφορά, ονομάζεται αισθητή θερμότητα.

Σε συστήματα που προορίζονται για computational sprinting, αυτά τα υλικά χρησιμοποιούνται λόγω της ιδιότητας τους να αποθηκεύουν θερμότητα υπό σταθερή περίπου θερμοκρασία. Αυτό έχει ως αποτέλεσμα, μία σταθερή και αξιόλογη ροή θερμότητας προς το υλικό αλλαγής φάσης, επειδή η ροή θερμότητας είναι γραμμικά εξαρτημένη από τη θερμοκρασιακή διαφορά μεταξύ δύο επιφανειών. Το τελικό αποτέλεσμα αυτού του γεγονότος, είναι ότι η χρήση τέτοιων υλικών σε ένα σύστημα, οδηγεί σε πιο αργές αυξήσεις θερμοκρασίας, δηλαδή σε μεγαλύτερα διαθέσιμα χρονικά παράθυρα υπολογισμού.

Όπως είναι αναμενόμενο, η λανθάνουσα θερμότητα τήξης ενός υλικού, ή, πιο σωστά, η ειδική θερμότητα τήξης, είναι παράγοντας που καθορίζει αν ένα υλικό είναι κατάλληλο για εφαρμογές sprinting, και τι επιπτώσεις θα έχει. Η ειδική θερμότητα τήξης είναι η ενέργεια που απαιτείται για την αλλαγή φάσης του υλικού από στερεό σε υγρό ανά μονάδα μάζας. Φυσικά, η ειδική θερμότητα τήξης είναι ιδιότητα του υλικού και είναι ανεξάρτητη του μεγέθους και των διαστάσεων ενός δείγματος.



Σχήμα 1: Συμπεριφορά υλικών αλλαγής φάσης και λανθάνουσα θερμότητα

Ένα άλλο σημαντικό χαρακτηριστικό για τα υλικά αλλαγής φάσης, είναι η θερμική αγωγιμότητα. Η θερμική αγωγιμότητα εκφράζει την ικανότητα ενός υλικού να μεταφέρει (άγει) θερμική ενέργεια. Εκφράζει το ρυθμό μεταφοράς θερμότητας. Η μεταφορά θερμότητας πραγματοποιείται με μικρότερο ρυθμό σε υλικά με μικρή θερμική αγωγιμότητα από ότι σε υλικά με μεγάλη. Αντίστοιχα, υλικά με μεγάλη θερμική αγωγιμότητα όπως ο χαλκός, χρησιμοποιούνται σε ψήκτρες επεξεργαστών ενώ υλικά με μικρή θερμική αγωγιμότητα χρησιμοποιούνται για θερμική μόνωση. Στην περίπτωση μας, η μεγάλη θερμική αγωγιμότητα είναι πολύ σημαντική, πράγμα που ισχύει για όλα τα υλικά που περιλαμβάνονται στη συσκευασία ενός ολοκληρωμένου. Επιπλέον, η ομοιογενής κατανομή θερμότητας, η οποία είναι άμεσο αποτέλεσμα υψηλής θερμικής αγωγιμότητας μετά από κάποιο χρονικό διάστημα, είναι ζωτικής σημασίας για να εκμεταλλευτούμε το μέγιστο από τη θερμική χωρητικότητα που προσφέρει ένα υλικό αλλαγής φάσης. Τυπικές τιμές θερμικής αγωγιμότητας για υλικά αλλαγής φάσης υποδεικνύουν τουλάχιστον τάξη μεγέθους διαφορά από τις αντίστοιχες τιμές για το χαλκό. Συνεπώς, η τοποθέτηση των υλικών αυτών στη στοίβα του ολοκληρωμένου και το πάχος των στρωμάτων που θα χρησιμοποιηθούν, χρήζουν προσεκτικής μελέτης.

Επιπροσθέτως, προσεκτική μελέτη πρέπει να δοθεί και στο σημείο τήξης του υλικού αλλαγής φάσης. Η ιδανική τιμή για ένα σύστημα που σκοπεύουμε να εφαρμόσουμε sprinting δεν είναι προφανής. Συνήθως, χρησιμοποιείται κάποια τιμή κοντά στις κρίσιμες θερμοκρασίες για το ολοκληρωμένο. Για να είμαστε πιο ακριβείς, η τιμή του σημείου τήξης επιλέγεται λίγο χαμηλότερη από την κρίσιμη θερμοκρασία για το σύστημα, έτσι ώστε να συνυπολογιστεί ένας βαθμός καθυστέρησης στην θέρμανση του στρώματος του υλικού αλλαγής φάσης. Γενικά, αποφεύγονται πολύ υψηλές θερμοκρασίες τήξης καθώς θέλουμε το υλικό να αλλάξει φάση πριν τα στοιχεία του ολοκληρωμένου φτάσουν σε κρίσιμες θερμοκρασίες. Αντίστοιχα, το ίδιο γίνεται και με πολύ χαμηλές θερμοκρασίες τήξης, διότι θα σπαταληθεί η ικανότητα των υλικών, να αποθηκεύουν μεγάλες ποσότητες ενέργειας κατά την αλλαγή φάσης, σε μη κρίσιμες θερμοκρασίες.

Τέλος, η ποσότητα υλικού που θα χρησιμοποιήσουμε αποτελεί ένα συμβιβασμό. Αυτό συμβαίνει καθώς, αυξάνοντας την ποσότητα του υλικού δεν αυξάνεται μόνο η διαθέσιμη θερμική χωρητικότητα, άλλα και η θερμική αντίσταση των ενεργών στοιχείων του ολοκληρωμένου (παράγουν θερμότητα) προς το περιβάλλον. Πρακτικά αυτό σημαίνει ότι το υλικό αλλαγής φάσης, παρόλο που αυξάνει την συνολική θερμική χωρητικότητα του συστήματος, δυσχεραίνει την αποβολή θερμότητας από αυτό. Με άλλα λόγια, εμποδίζει την απρόσκοπτη ψύξη του συστήματος με φυσικά μέσα.

Συνολικά, παρόλο που η χρήση υλικών αλλαγής φάσης σε υπολογιστικά συστήματα, συνδυαστικά με computational sprinting υπάρχει ήδη στην βιβλιογραφία, σύμφωνα με την ανάλυση που παρουσιάσαμε, υπάρχει ακόμα χώρος για διερεύνηση. Πιο συγκεκριμένα, εφαρμογές είτε σε πρακτικό είτε σε θεωρητικό επίπεδο, κυρίως αποδεικνύουν την ικανότητα συγκεκριμένων υλικών να υποβοηθήσουν διάφορες μεθοδολογίες sprinting. Στη δικιά μας περίπτωση, στρεφόμαστε πιο πολύ στην αναζήτηση των επιθυμητών χαρακτηριστικών των

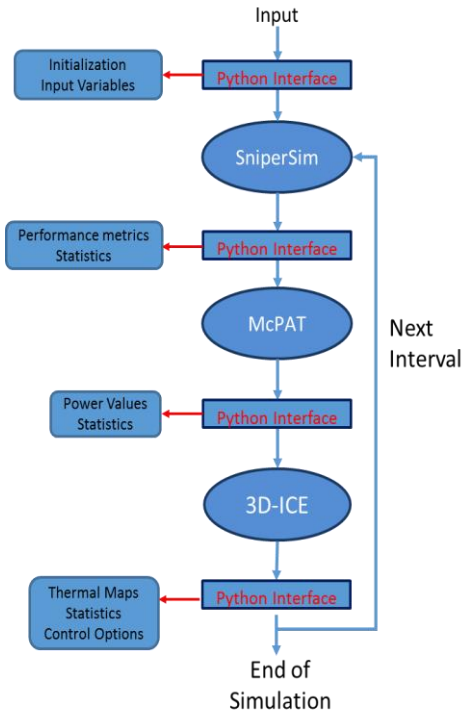
υλικών αλλαγής φάσης και στην αναζήτηση τρόπων για την περαιτέρω αύξηση της αποδοτικότητας τους.

Το πρώτο βήμα προς αυτή την κατεύθυνση, ήταν η κατασκευή ενός μοντέλου προσομοίωσης. Το μοντέλο προσομοίωσης που κατασκευάσαμε, αποτελείται από ένα πρόγραμμα προσομοίωσης υλικού (Sniper Simulator), ένα πρόγραμμα προσομοίωσης κατανάλωσης ισχύος (McPAT), ένα πρόγραμμα θερμικής προσομοίωσης (3D-ICE) και ένα σύνολο από υποπρογράμματα γραμμένα σε Python τα οποία πραγματοποιούν ένα σύνολο δραστηριοτήτων σχετικές με τη λειτουργία του πλαισίου προσομοίωσης που κατασκευάσαμε. Τα υποπρογράμματα αυτά ονομάζουμε για χάρη σαφήνειας πλαίσιο διασύνδεσης. Οι δραστηριότητες που επιτελεί το πλαίσιο διασύνδεσης, περιλαμβάνουν την συγκέντρωση των χαρακτηριστικών της εκάστοτε προσομοίωσης, σαν είσοδο από τον χρήστη, και τη σωστή εισαγωγή τους στα επιμέρους προγράμματα, την αρχικοποίηση, την έναρξη και τον τερματισμό κάθε προσομοιωτή, τη διασύνδεση μεταξύ τους, την εξαγωγή στατιστικών και άλλων αρχείων και την αποθήκευση τους για περαιτέρω επεξεργασία.

Το πλαίσιο που περιγράφηκε, προσομοιώνει τη λειτουργία επεξεργαστικών μονάδων από πλευράς επιδόσεων, κατανάλωσης ισχύος και θερμικής συμπεριφοράς. Με βάση τα προγράμματα που χρησιμοποιήσαμε για την κατασκευή του πλαισίου, το υλικό που μπορεί να προσομοιωθεί έχει μεγάλη ποικιλία και περιλαμβάνει πολυπύρηνες πλατφόρμες με εκατοντάδες πυρήνες και διαφοροποιήσιμα αρχιτεκτονικά χαρακτηριστικά.

Σημαντικό χαρακτηριστικό του πλαισίου προσομοίωσης που κατασκευάσαμε, είναι το γεγονός ότι η λειτουργία του πραγματοποιείται σε κύκλους-βήματα, συγκεκριμένου χρονικού διαστήματος, το οποίο καθορίζεται από το χρήστη. Το χαρακτηριστικό αυτό μας επιτρέπει να παρατηρούμε τη συμπεριφορά του συστήματος στο τέλος κάθε κύκλου, να συγκεντρώνουμε στατιστικά στοιχεία ανά τακτά χρονικά διαστήματα και να επιτελούμε ενέργειες ελέγχου αν κριθεί απαραίτητο.

Η βασική ροή που ακολουθείται κατά τη διάρκεια μιας προσομοίωσης παρουσιάζεται σύντομα στο Σχήμα 2. Όπως φαίνεται στο σχήμα, πριν από την κλήση καθενός από τα προγράμματα προσομοίωσης, προηγείται μια κλήση στο πλαίσιο διασύνδεσης (το σύνολο προγραμμάτων στη γλώσσα Python). Γενικά, οι κλήσεις αυτές στο πλαίσιο διασύνδεσης αφορούν κυρίως την εξαγωγή στατιστικών σχετικών με την έξοδο του προηγούμενου επιπέδου προσομοίωσης, την συγκέντρωση των εισόδων για το επόμενο επίπεδο, την διαμόρφωση τους σε κατάλληλη μορφή και την κλήση του επόμενου επιπέδου.



Σχήμα 2: Βασική ροή πλαισίου προσομοίωσης

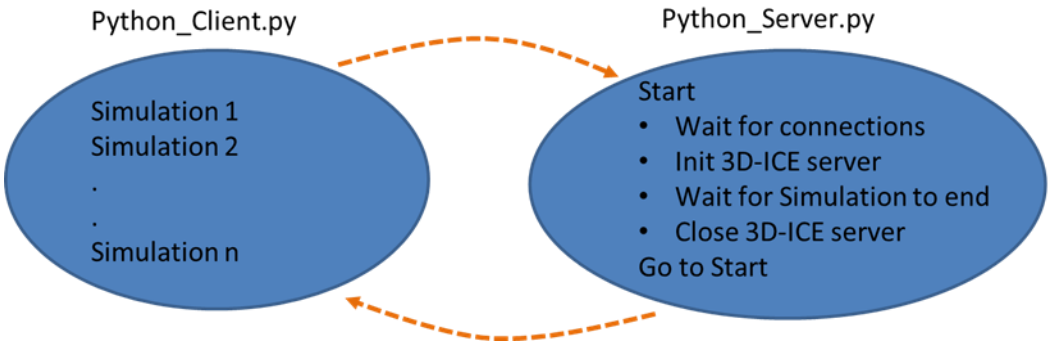
Για να μπορεί το πλαίσιο που φτιάξαμε να εξυπηρετεί μια ουρά προσομοιώσεων, και να μην είναι απαραίτητη η εισαγωγή καινούριων στοιχείων από το χρήστη κάθε φορά που τελειώνει ενα σενάριο προσομοίωσης, δημιουργήσαμε ένα μοντέλο πελάτη – εξυπηρετητή για το πλαίσιο διασύνδεσης. Παράλληλα, δημιουργήσαμε και μία υποενότητα του πλαισίου διασύνδεσης στο οποίο μπορεί ο χρήστης να προσδιορίζει τα χαρακτηριστικά των προσομοιώσεων που θέλει να εκτελεστούν. Τα χαρακτηριστικά αυτά, προσδιορίζονται στο αρχείο Simulation_Queue.py που είναι και το μόνο αρχείο το οποίο χρειάζεται να εισάγει στοιχεία ο χρήστης και να το καλέσει. Μετά την κλήση αυτή, το πλαίσιο διασύνδεσης αναλαμβάνει όλες τις απαραίτητες ενέργειες μέχρι το πέρας όλων των ζητούμενων σεναρίων προσομοίωσης.

Ένας από τους λόγους που οδήγησαν στη διαμόρφωση του πλαισίου διασύνδεσης με αυτό τον τρόπο, είναι η εγγενής λειτουργία του προγράμματος θερμικής προσομοίωσης με αντίστοιχο μοντέλο πελάτη – εξυπηρετητή. Στην περίπτωση του 3D-ICE, η πλευρά του εξυπηρετητή περιέχει όλες τις δομές και επιτελεί όλες τις απαραίτητες ενέργειες για την προσομοίωση. Η πλευρά του πελάτη αποτελεί απλά ένα ενδιάμεσο πλαίσιο επικοινωνίας. Γενικά, μια εκτέλεση του πελάτη του 3D-ICE αντιστοιχεί σε ενα χρονικό κύκλο προσομοίωσης, ενώ, μια εκτέλεση του εξυπηρετητή του 3D-ICE, αντιστοιχεί σε ένα ολόκληρο σενάριο προσομοίωσης.

Στο πλαίσιο που δημιουργήσαμε, η πλευρά του εξυπηρετητή περιμένει να δοθεί σήμα ότι ζητήθηκε κάποιο σύνολο σεναρίων προσομοίωσης και εκκινεί τον εξυπηρετητή του 3D-ICE με τις κατάλληλες παραμέτρους. Όταν τελειώσει ένα σενάριο προσομοίωσης, τερματίζει τον εξυπηρετητή του 3D-ICE και περιμένει για νέες συνδέσεις. Στην περίπτωση που έχουμε ζητήσει πολλά σενάρια, η επόμενη σύνδεση θα γίνει αυτόματα μόλις το προηγούμενο σενάριο

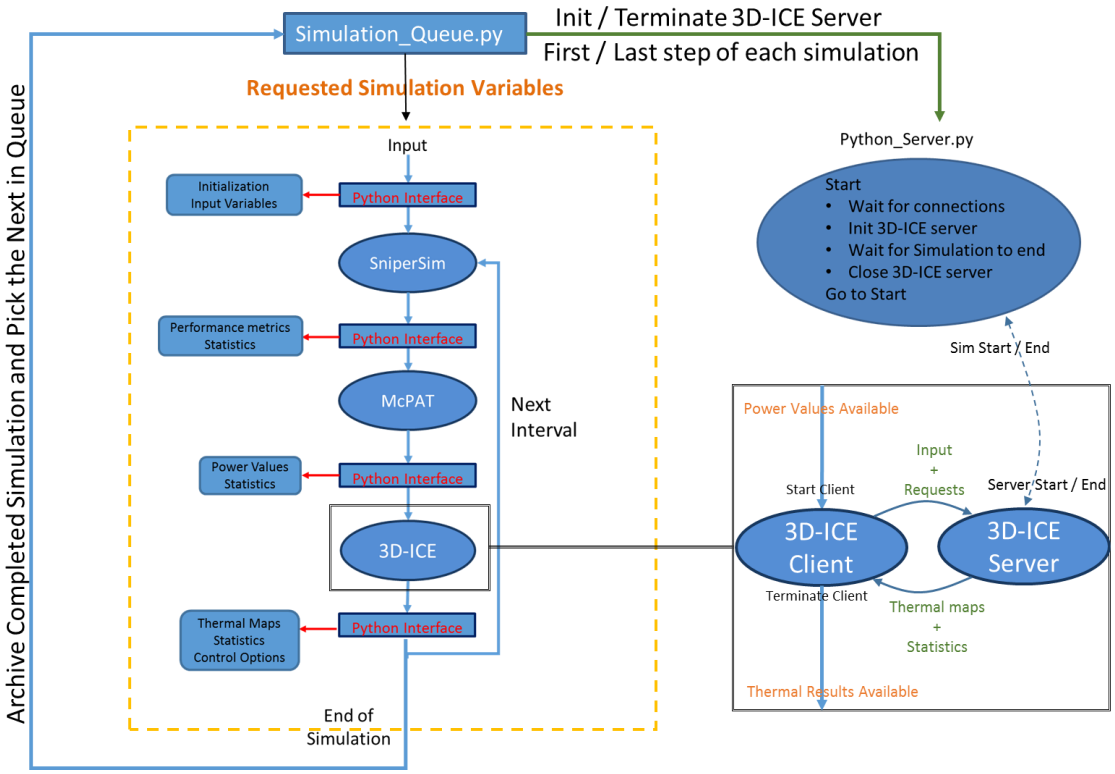
ολοκληρώσει όλες τις διαδικασίες που του αναλογούν. Ο αντίστοιχος πελάτης του πλαισίου διασύνδεσης, επιτρέπει στο χρήστη να προσδιορίσει μια ουρά από ζητούμενα σενάρια προσομοίωσης και διευκολύνει την εισαγωγή παραμέτρων για όλα τα εργαλεία προσομοίωσης σε ένα ενοποιημένο πλαίσιο. Το υψηλότερο επίπεδο του πλαισίου αυτού, είναι το αρχείο Simulation_Queue.py που αναφέραμε προηγουμένως.

Έτσι, ο εξυπηρετητής σε Python χρειάζεται να εκκινηθεί μία και μόνο φορά, και έπειτα εξυπηρετεί αιτήματα προσομοίωσης επ' αόριστο. Σε διαφορετική περίπτωση, θα χρειαζόταν αφού τελειώσει κάθε σενάριο προσομοίωσης, να εκκινούμε εκ νέου τον εξυπηρετητή του 3D-ICE με τις κατάλληλες εισόδους και μετά να εκκινούμε το Sniper Simulator με τις δικές του παραμέτρους. Ενα απλουστευμένο μοντέλο του πελάτη – εξυπηρετητή σε Python, φαίνεται στο Σχήμα 3.



Σχήμα 3: Μοντέλο πελάτη-εξυπηρετητή σε Python

Για να είναι πιο σαφές, το πως λειτουργεί παράλληλα, το μοντέλο σε κύκλους μίας προσομοίωσης, με το μοντέλο πελάτη – εξυπηρετητή του πλαισίου διασύνδεσης, διαμορφώσαμε το Σχήμα 4. Στο σχήμα αυτό βλέπουμε σαν πρώτο βήμα το αρχείο Simulation_Queue.py. Το αρχείο αυτό, όταν κληθεί, δίνει σήμα στον εξυπηρετητή του πλαισίου διασύνδεσης (Python_Server.py) να εκκινήσει τον εξυπηρετητή του 3D-ICE, γεγονός που μεταφράζεται ως η έναρξη μιας νέας προσομοίωσης.



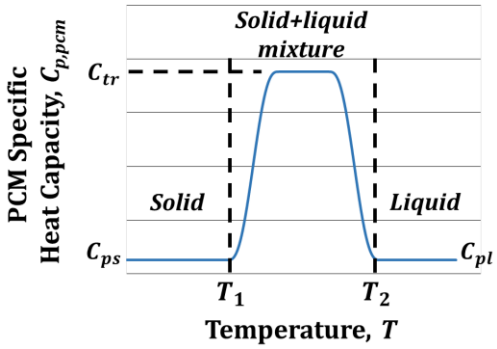
Σχήμα 4: Σύνοψη του συνολικού πλαισίου προσομοιώσεων

Στη συνέχεια, ξεκινάει η διαδικασία προσομοίωσης σε βήματα, όπου πρώτα ολοκληρώνει ενα κύκλο το Sniper Simulator παράγοντας δεδομένα σχετικά με την επίδοση και τη λειτουργία του υλικού που προσομοιώνουμε. Από την έξοδο του τροφοδοτείται το McPAT και παράγει δεδομένα κατανάλωσης ισχύος, τα οποία εισάγονται στον πελάτη του 3D-ICE. Ο πελάτης του 3D-ICE μεταφέρει τα δεδομένα αυτά μαζί με τις απαιτούμενες ενέργειες στον εξυπηρετητή του 3D-ICE και περιμένει το αποτέλεσμα της προσομοίωσης το οποίο καταγράφει. Στη συνέχεια ένας καινούριος κύκλος ξεκινάει και η διαδικασία αυτή συνεχίζεται μέχρι το τέλος του σεναρίου προσομοίωσης. Έπειτα, η ίδια διαδικασία ξεκινάει εκ νέου για ένα νέο σενάριο προσομοίωσης μέχρις ότου ολοκληρωθούν όλα τα σενάρια που έχουμε προσδιορίσει.

Όσον αφορά τα εργαλεία που χρησιμοποιήσαμε για να δομήσουμε όλο αυτό το σύνολο που μόλις περιγράψαμε, στην περίπτωση του McPAT και του Sniper Simulator, χρειάστηκε να τα τροποποιήσουμε ελάχιστα για να επιτελούν τις λειτουργίες που θέλαμε και με τον επιθυμητό τρόπο. Επειδή είναι προγράμματα που επιδέχονται μεγάλη προσαρμογή μέσα από το πέρασμα παραμέτρων, ο βασικός όγκος εξατομίκευσης τους πραγματοποιήθηκε χρησιμοποιώντας κατάλληλες εισόδους μέσω του πλαισίου διασύνδεσης.

Στην περίπτωση του 3D-ICE όμως, χρειάστηκε να τροποποιήσουμε εκτενώς τον πηγαίο κώδικα του προγράμματος, για να μπορέσουμε να προσομοιώσουμε τα ζητούμενα υλικά αλλαγής φάσης. Συγκεκριμένα, για να το επιτύχουμε αυτό, χρησιμοποιήσαμε μια τεχνική που περιγράφεται στο [9] και περιγράφεται με τον όρο *apparent heat capacity method*. Η τεχνική αυτή, αποδίδει μια μη-γραμμική τιμή στη θερμική χωρητικότητα του υλικού, η οποία είναι συνάρτηση της

θερμοκρασίας και έχει τη μορφή που φαίνεται στο Σχήμα 5. Με τη μέθοδο αυτή, η μετάβαση του υλικού αλλαγής φάσης από στερεό σε υγρό, πραγματοποιείται σε ένα διάστημα θερμοκρασιών όπου η θερμική χωρητικότητα του υλικού είναι πολύ μεγάλη σε σύγκριση με αυτή της στερεάς και υγρής φάσης. Εξαιτίας της μεγάλης αυτής αύξησης της θερμικής χωρητικότητας, ο ρυθμός μεταβολής της θερμοκρασίας μειώνεται κατά πολύ κατά τη διάρκεια αλλαγής φάσης.



Σχήμα 5: Συνάρτηση θερμικής χωρητικότητας για υλικά αλλαγής φάσης

Σε πραγματικές συνθήκες, κατά την αλλαγή φάσης, το υλικό απορροφά μεγάλα ποσά ενέργειας σε περίπου σταθερή θερμοκρασία. Η συμπεριφορά που προκύπτει αλλάζοντας τη θερμική χωρητικότητα του υλικού δυναμικά κατά τη διάρκεια της προσομοίωσης, προσεγγίζει πραγματικές αλλαγές φάσης πάρα πολύ καλά.

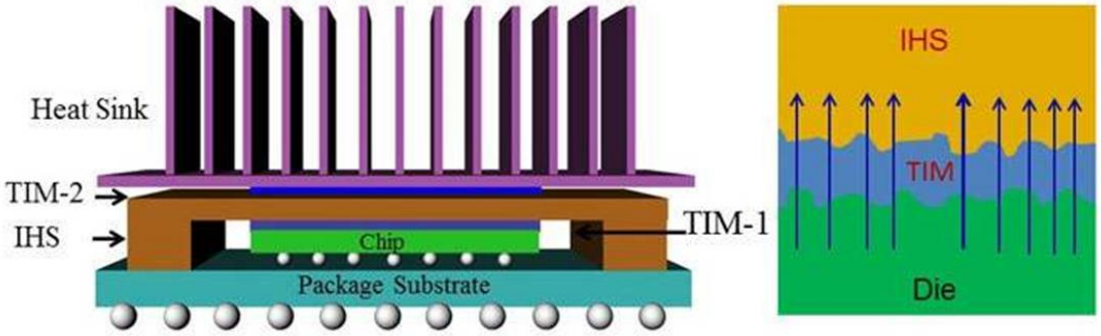
Για να εφαρμόσουμε τη μέθοδο αυτή στο 3D-ICE, εισάγαμε σαν είσοδο άλλο ένα αρχείο που περιγράφει τα χαρακτηριστικά του υλικού αλλαγή φάσης. Στη συνέχεια, προσθέσαμε μια συνάρτηση που επεξεργάζεται το περιεχόμενο αυτού του αρχείου και το αποθηκεύει σε κατάλληλες δομές. Τέλος, τροποποιήσαμε το πρόγραμμα, ώστε να ελέγχει όταν κάνει ανάθεση θερμικών χωρητικοτήτων, αν το υλικό έχει προσδιοριστεί ως υλικό αλλαγής φάσης και αν βρίσκεται στο θερμοκρασιακό διάστημα που μας ενδιαφέρει. Τότε και μόνο τότε, το πρόγραμμα θα αναθέσει την διαφορετική θερμική χωρητικότητα και θα συνεχίσει την κανονική λειτουργία του.

Είναι εμφανές, ότι η παρουσίαση των αλλαγών που χρειάστηκε να κάνουμε στο πρόγραμμα είναι συνοπτική και γενικά μη-αναλυτική. Ο σκοπός της προσέγγισης αυτής είναι να τονιστεί η λογική που ακολουθήσαμε και όχι ο συγκεκριμένος κώδικας και άλλα τεχνικά ζητήματα. Επιπροσθέτως, άλλες πιο μικρές αλλαγές που χρειάστηκε να γίνουν όπως π.χ. προσθήκη βοηθητικών μεταβλητών, συναρτήσεων και δομών δεν περιγράφονται για χάρη απλότητας.

Η πρώτη εφαρμογή του πλαισίου προσομοίωσης μας, στρέφεται στο να τονίσει τη σημασία των υλικών διεπαφής και της κατανομής ισχύος και κατ' επέκταση, θερμότητας, στην επιφάνεια ενός ολοκληρωμένου. Οι δύο παράγοντες αυτοί, εξετάστηκαν καθώς σε πολλές περιπτώσεις στη βιβλιογραφία, παραλείπονται, τη στιγμή που, για το είδος το συστημάτων που θέλουμε να εξετάσουμε, θεωρούμε ότι έχουν αυξημένη επίδραση στην ακρίβεια των αποτελεσμάτων. Για το

λόγο αυτό, επιλέξαμε να δοκιμάσουμε αυτή τη θεωρία, να διαμορφώσουμε κατάλληλα το θερμικό μας μοντέλο, και έπειτα να προχωρήσουμε σε προσομοιώσεις με υλικά αλλαγής φάσης.

Τα υλικά διεπαφής, χρησιμοποιούνται για να ενώσουμε τις επιφάνειες μεταξύ της συσκευασίας ενός ολοκληρωμένου και του χάλκινου καλύμματος που χρησιμοποιείται για την απαγωγή θερμότητας. Οι επιφάνειες μεταξύ των δύο υλικών, χαρακτηρίζονται, σε μικροσκοπικό επίπεδο, από ανωμαλίες οι οποίες δημιουργούν κενά αέρος. Τα υλικά διεπαφής χρησιμοποιούνται και για τη συγκόλληση των επιφανειών αλλά και για την εξάλειψη αυτών των κενών αέρος. Τα υλικά αυτά, επειδή έχουν μεγαλύτερη θερμική αγωγιμότητα από τον αέρα που αντικαθιστούν, έχουν ως αποτέλεσμα τη μείωση της θερμικής αντίστασης της διεπαφής. Όπως είναι αναμενόμενο, μείωση της θερμικής αντίστασης είναι επιθυμητή για την καλύτερη απαγωγή θερμότητας. Διαφορετικά υλικά διεπαφής χρησιμοποιούνται με ανάλογο τρόπο και για την τοποθέτηση της ψήκτρας, αν υπάρχει, στο χάλκινο κάλυμμα. Η εφαρμογή υλικών διεπαφής καθώς και ένα από τα πιο διαδεδομένα μοντέλα συσκευασίας ενός ολοκληρωμένου παρουσιάζονται στο Σχήμα 6.



Σχήμα 6 : (α) Δομικά στοιχεία ενός ολοκληρωμένου
(β) Γραφική αναπαράσταση εφαρμογής υλικού διεπαφής

Σχετικά με τις κατανομές θερμότητας και ισχύος κατά μήκος των ενεργών στοιχείων ενός ολοκληρωμένου, έχει αποδειχθεί στη βιβλιογραφία ότι η κατανομή αυτή εξαρτάται από την αρχιτεκτονική ή/και το είδος των εργασιών που επιτελούνται. Αυτό το γεγονός τονίζεται διότι η θερμική συμπεριφορά των ολοκληρωμένων είναι αρκετά διαφορετική όταν έχουμε ομοιόμορφη κατανομή θερμότητας και όταν έχουμε συγκεντρώσεις θερμών σημείων. Παράλληλα, για χάρη απλότητας, σε αρκετές έρευνες χρησιμοποιείται ομοιόμορφη κατανομή παρά το γεγονός ότι μια τέτοια κατανομή απέχει από πραγματικά δεδομένα.

Σε συστήματα που εφαρμόζουν sprinting, πιστεύουμε ότι η ανισοκατανομή αυτή θα είναι ακόμα πιο έντονη, ως αποτέλεσμα, όχι μόνο υψηλότερων συχνοτήτων στις επεξεργαστικές μονάδες, αλλά και διαφορετικής χρησιμοποίησης των διαθέσιμων πυρήνων. Επιπροσθέτως, η παρουσία υλικών αλλαγής φάσης στο σύστημα προσθέτει άλλο ένα παράγοντα ανακρίβειας, καθώς, με ομοιόμορφες κατανομές θερμότητας, αδυνατούμε να προσομοιώσουμε το τοπικό λιώσιμο του υλικού στις πιο θερμές περιοχές που δημιουργούνται σε πραγματικές συνθήκες.

Τα υλικά διεπαφής που αναφέραμε προηγουμένως, στη γενική περίπτωση παραλείπονται και αυτά από τα θερμικά μοντέλα, γεγονός που παραμορφώνει ακόμα περισσότερο το θερμικό

προφίλ της επεξεργαστικής μονάδας μας. Η παραμόρφωση αυτή επηρεάζεται και λειτουργεί αθροιστικά με τους παράγοντες που αναφέραμε προηγουμένως.

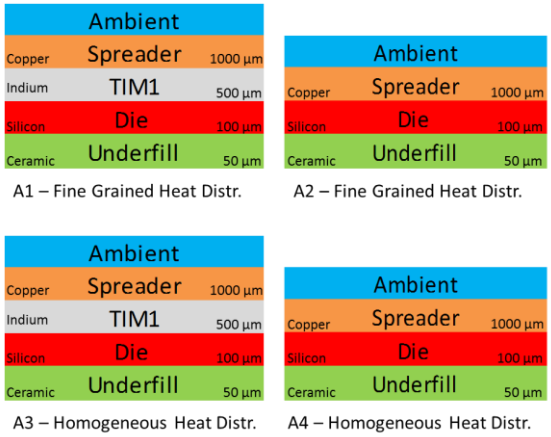
Για να αξιολογήσουμε την επίδραση των υλικών διεπαφής και της κατανομής θερμότητας σε συστήματα που χρησιμοποιούν συχνότητες λειτουργίας μεγαλύτερες από τις προδιαγραφές ασφαλούς λειτουργίας του ολοκληρωμένου, διαμορφώσαμε ένα σύνολο προσομοιώσεων. Στις προσομοιώσεις αυτές, συγκρίναμε για τρία διαφορετικά προγράμματα, την συμπεριφορά της επεξεργαστικής μονάδας, σε τέσσερις διαφορετικές περιπτώσεις. Όταν συμπεριλαμβάνουμε τα υλικά διεπαφής και την άνιση κατανομή θερμότητας, όταν παραλείπουμε τα υλικά διεπαφής μόνο, όταν υποθέτουμε ομοιόμορφη κατανομή θερμότητας μόνο, και όταν συνδυάζουμε και τους δύο παράγοντες ανακρίβειας ταυτόχρονα.

Η επεξεργαστική μονάδα που επιλέξαμε να προσομοιώσουμε, ήταν ένας τετραπύρηνος επεξεργαστής βασισμένος στο μοντέλο Nehalem Gainestown. Για να προσομοιώσουμε καλύτερα μεγάλες πυκνότητες ισχύος, συμβατές με τις σύγχρονες επεξεργαστικές μονάδες, επιλέξαμε να προσαρμόσουμε το μοντέλο μας στην κλίμακα των 22nm, βρίσκοντας τα μεγέθη και το εμβαδό των στοιχείων από το McPAT. Η τιμή που λάβαμε για το εμβαδό του ολοκληρωμένου ήταν 48,36 mm^2 το οποίο προσομοιώσαμε ως ένα 8 mm x 8 mm τετραγωνικό τσιπ, υποθέτοντας 20% σφάλμα που ήταν η μέση υποεκτίμηση που παρουσιάστηκε στο [18] για το McPAT.

Για την προσομοίωση αυτή, χρησιμοποιήθηκαν χρονικά βήματα των 1ms. Οι σουίτες προγραμμάτων που χρησιμοποιήθηκαν ήταν Blackscholes, Bodytrack και Streamcluster, όλες για συνολικά 4 νήματα και για 1000ms συνολική διάρκεια προσομοίωσης. Επειδή στην περίπτωση που εξετάζουμε, δεν σκοπεύουμε να εφαρμόσουμε δυναμικό έλεγχο πάνω σε χαρακτηριστικά του Sniper Simulator, για να επιταχύνουμε το χρόνο εκτέλεσης κάθε προσομοίωσης, καταγράψαμε τις τιμές ισχύων των ενεργών στοιχείων χρησιμοποιώντας το πλαίσιο που κατασκευάσαμε και τις τροφοδοτήσαμε κατευθείαν στο 3D-ICE παρακάμπτοντας τα άλλα δύο εργαλεία. Η λογική πίσω από αυτή την ενέργεια, έχει να κάνει με το γεγονός ότι χρησιμοποιούσαμε τα ίδια σενάρια με αλλαγές μόνο στο θερμικό μοντέλο. Κάθε πρόγραμμα, προσομοιώθηκε με όλους τους πυρήνες να λειτουργούν στα 2190, 2390, 2660 και 2926MHz.

Όσον αφορά το θερμικό μοντέλο, επιλέξαμε να χρησιμοποιήσουμε τη διάταξη που είδαμε στο Σχήμα 6 χωρίς όμως την ψήκτρα. Αυτή η απόφαση επηρεάστηκε από το [10] που χρησιμοποιείται ένα αντίστοιχο μοντέλο. Μια συνοπτική παρουσίαση της στοίβας που αποτελεί το ολοκληρωμένο που χρησιμοποιήσαμε, φαίνεται στο Σχήμα 7 μαζί με τα υλικό και το πάχος κάθε στρώματος.

Στο 3D-ICE, χρησιμοποιήσαμε δομικές μονάδες μήκους και πλάτους 100μm. Το ύψος κάθε στρώματος ήταν κατά μέγιστο 250μm. Στα σενάρια προσομοιώσεων που διεξάγαμε όλα τα υλικά αρχικοποιούνται σε θερμοκρασία λίγο μεγαλύτερη από θερμοκρασία δωματίου, δηλαδή 300K. Η τιμή αυτή είναι συνεπής για επεξεργαστικές μονάδες που βρίσκονται σε λειτουργία αλλά δεν επιτελούν κάποια εργασία.



Σχήμα 7 : Διατάξεις που χρησιμοποιήθηκαν στις προσομοιώσεις

Για να δοκιμάσουμε τη θεωρία που αναφέραμε προηγουμένως, δημιουργήσαμε τέσσερις διατάξεις όπως φαίνεται στο Σχήμα 7. Η πρώτη παρουσιάζει το μοντέλο που θεωρούμε ότι είναι το πλέον ακριβές. Η δεύτερη παραλείπει τα υλικά διεπαφής. Η τρίτη διάταξη περιέχει τα υλικά διεπαφής αλλά υποθέτει ομοιογενή κατανομή θερμότητας, και η τελευταία παραλείπει τα υλικά διεπαφής θεωρώντας παράλληλα ομοιογενή κατανομή θερμότητας. Οι διατάξεις αυτές προσομοιώθηκαν για όλα τα επίπεδα συχνότητας λειτουργίας που αναφέρθηκαν προηγουμένως. Αξίζει να σημειωθεί ότι για τα προγράμματα που χρησιμοποιήσαμε, το γεγονός ότι αναθέσαμε 4 νήματα σε κάθε ένα, δεν σημαίνει ότι 4 νήματα είναι ενεργά καθόλη τη διάρκεια της προσομοίωσης. Αντιθέτως, σπάνια θα λειτουργούν και τα 4 νήματα στον ίδιο βαθμό. Αυτό είναι ιδανικό στην περίπτωση μας γιατί δημιουργεί ποικιλομορφία όχι μόνο στο βαθμό χρησιμοποίησης των στοιχείων ενός πυρήνα, άλλα και στο βαθμό χρησιμοποίησης των πυρήνων μεταξύ τους.

Όπως φαίνεται στο Σχήμα 7, κάθε μία από τις διατάξεις που χρησιμοποιήσαμε έχει ονομαστεί A1-A4. Η διάταξη A1 είναι αυτή που θεωρούμε ως την πιο ολοκληρωμένη. Οι άλλες διατάξεις, θεωρούμε ότι περιέχουν ανακρίβειες οι οποίες εντείνονται με την αύξηση της συχνότητας λειτουργίας των πυρήνων. Στο πλαίσιο αυτό, συγκρίναμε κάθε μία από τις διατάξεις A2-A4 με τη δικιά μας (A1), και συνοψίσαμε τα αποτελέσματα στον Πίνακα 1. Σημειώνεται, ότι οι τιμές μέγιστου σφάλματος και μέσου όρου σφάλματος που παρουσιάζονται στον Πίνακα 1, αφορούν τη μέγιστη παρατηρούμενη θερμοκρασία στο ολοκληρωμένο.

Τα στοιχεία του Πίνακα 1 επιβεβαιώνουν την σχέση μεταξύ συχνότητας λειτουργίας και θερμοκρασιακής διαφοράς μεταξύ των προσομοιώσεων. Η σχέση αυτή, είναι εμφανής τόσο για τη μέγιστη τιμή του θερμοκρασιακού σφάλματος όσο και για το μέσο όρο. Επίσης, το συμπέρασμα αυτό είναι εμφανές για όλα τα διαφορετικά προγράμματα που δοκιμάσαμε. Αξιοσημείωτο είναι το γεγονός ότι για το Bodytrack, όλες οι μετρικές σφάλματος ήταν αισθητά μεγαλύτερες, γεγονός που αναμέναμε λόγω της μεγαλύτερης ανομοιογένειας στη χρησιμοποίηση των διαθέσιμων πυρήνων.

Επιπροσθέτως, από την παρατήρηση των δεδομένων του πίνακα, φαίνεται ότι η ανομοιογένεια στην κατανομή θερμότητας, οδηγεί σε μεγαλύτερα σφάλματα σε σχέση με τα υλικά

διεπαφής. Ακόμη, επιβεβαιώνεται και άλλη μια προσδοκία μας, πως οι δύο ανακρίβειες λειτουργούν αθροιστικά (A1 – A4) και οδηγούν σε ακόμα μεγαλύτερα σφάλματα.

	MHz\ C°	Maximum temperature error			Average temperature error		
Blackscholes	Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
	2120	2.80	3.61	3.87	0.82	3.34	2.15
	2390	3.72	4.64	5.00	1.16	4.28	2.85
	2660	5.21	8.10	8.15	2.83	7.28	5.46
	2926	7.61	11.28	11.60	4.30	10.22	7.91
	MHz\ C°	Maximum temperature error			Average temperature error		
Bodytrack	Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
	2120	5.13	5.57	7.21	1.26	3.63	2.65
	2390	7.22	8.35	10.19	1.89	4.67	3.60
	2660	11.48	14.87	18.15	3.83	8.05	6.81
	2926	16.33	20.32	24.93	5.73	11.37	9.94
	MHz\ C°	Maximum temperature error			Average temperature error		
Streamcluster	Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
	2120	3.03	3.12	4.30	0.88	2.89	1.27
	2390	4.01	4.06	5.57	1.14	3.58	1.55
	2660	6.89	8.52	10.10	2.13	7.52	4.78
	2926	10.07	12.45	14.80	3.45	10.90	7.27

Πίνακας 1: Μέσος όρος και μέγιστη τιμή θερμοκρασιακού σφάλματος μεταξύ των διαφορετικών διατάξεων

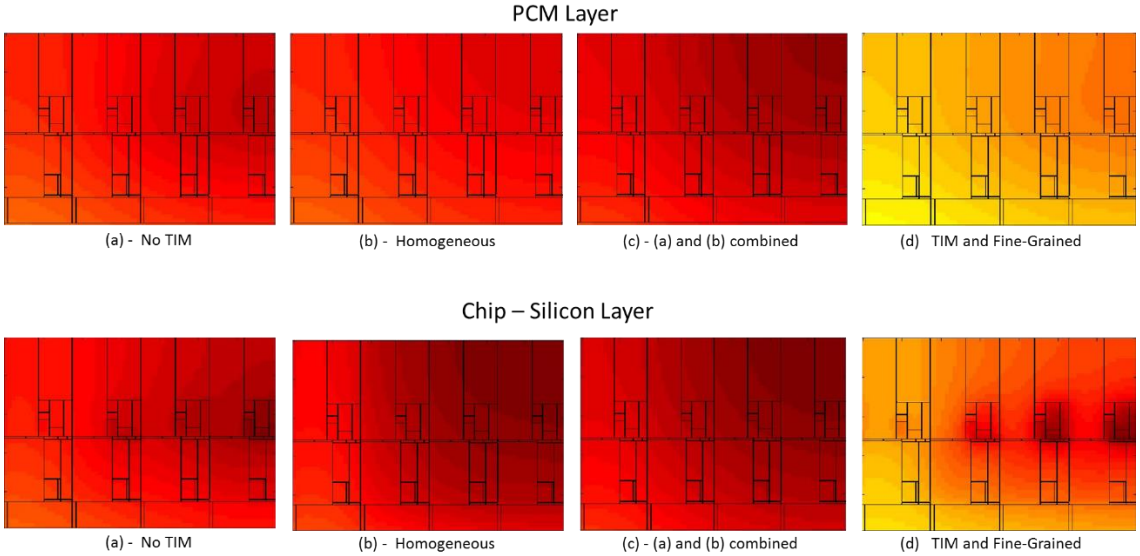
Τα σφάλματα αυτά, όπως προαναφέρθηκε, αφορούν τη μέγιστη θερμοκρασία στο ολοκληρωμένο και όπως είδαμε μπορούν να πάρουν μεγάλες τιμές. Το γεγονός αυτό χρήζει ιδιαίτερης σημασίας καθώς η μέγιστη θερμοκρασία του ολοκληρωμένου αποτελεί μεταβλητή ελέγχου για συστήματα που εφαρμόζουν sprinting και επηρεάζει άμεσα την επίδραση των υλικών αλλαγής φάσης.

Για να δείξουμε πιο παραστατικά ένα παράδειγμα της επίδρασης των παραπάνω παραγόντων, πραγματοποιήσαμε άλλη μία ομάδα προσομοιώσεων όπου προσθέσαμε επιπλέον και ένα λεπτό στρώμα υλικού αλλαγής φάσης πάνω από το χάλκινο κάλυμμα του ολοκληρωμένου. Το σημείο τήξης του υλικού τέθηκε στους 60 βαθμούς Κελσίου και οι προσομοιώσεις έγιναν για το πρόγραμμα Blackscholes σε συχνότητα λειτουργίας 2920 MHz.

Στις προσομοιώσεις αυτές, χρησιμοποιήθηκαν οι διατάξεις που περιγράφηκαν προηγουμένως, δηλαδή, με και χωρίς υλικά διεπαφής, με ομοιογενή και ανομοιογενή κατανομή θερμότητας.

Τα αποτελέσματα των προσομοιώσεων, έδειξαν για το θερμικό μοντέλο που θεωρούμε πιο ακριβές, δηλαδή με υλικά διεπαφής και ανομοιογενή κατανομή θερμότητας, 718 ms μέχρι την πρώτη παραβίαση των θερμικών ορίων. Για την περίπτωση που αγνοούνται μόνο τα υλικά διεπαφής, ο αντίστοιχος χρόνος ήταν 781 ms. Θεωρώντας μόνο ομοιογενή κατανομή θερμότητας αλλά χρησιμοποιώντας υλικά διεπαφής, λάβαμε χρόνο προσομοίωσης 908 ms, ενώ, θεωρώντας ομοιόμορφη κατανομή θερμότητας και αγνοώντας τα υλικά διεπαφής, ο αντίστοιχος χρόνος ήταν 847 ms.

Παρόλο που οι αποκλίσεις που αναφέραμε είναι σημαντικές, για να γίνει περισσότερο κατανοητό το πόσο διαφορετικά συμπεριφέρεται το σύστημα σε κάθε περίπτωση, δημιουργήσαμε το Σχήμα 8. Στο σχήμα φαίνονται στην πρώτη γραμμή οι θερμικές εικόνες για το στρώμα του υλικού αλλαγής φάσης και στη δεύτερη για το στρώμα του πυρήνα. Αρκεί να συγκρίνει κανείς την τελευταία εικόνα κάθε γραμμής, σε σχέση με τις προηγούμενες για να καταλάβει πόσο διαφέρουν τα θερμικά μοντέλα μεταξύ τους.



Σχήμα 8: Θερμικές εικόνες για κάθε διάταξη τη στιγμή της πρώτης παραβίασης των θερμοκρασιακών ορίων

Έτσι, βλέπουμε ότι οι δύο παράγοντες που εξετάσαμε, είναι ιδιαίτερα σημαντικοί στη διερεύνηση της χρήσης των υλικών αλλαγής φάσης, ειδικά σε μεγάλες συχνότητες λειτουργίας και παρά το γεγονός ότι έχουν αρνητική επίδραση στους χρόνους προσομοίωσης (χρειάζεται να προσομοιώσουμε περισσότερα δεδομένα).

Στη συνέχεια, αφού διαμορφώσαμε πλέον πλήρως το θερμικό μοντέλο που σκοπεύουμε να χρησιμοποιήσουμε, με γνώμονα την ακρίβεια αλλά και ανεκτούς χρόνους προσομοίωσης, διεξάγαμε μία σειρά από προσομοιώσεις, με σκοπό τη διερεύνηση διαφορετικών διατάξεων επεξεργαστικών μονάδων που κάνουν χρήση υλικών αλλαγής φάσης.

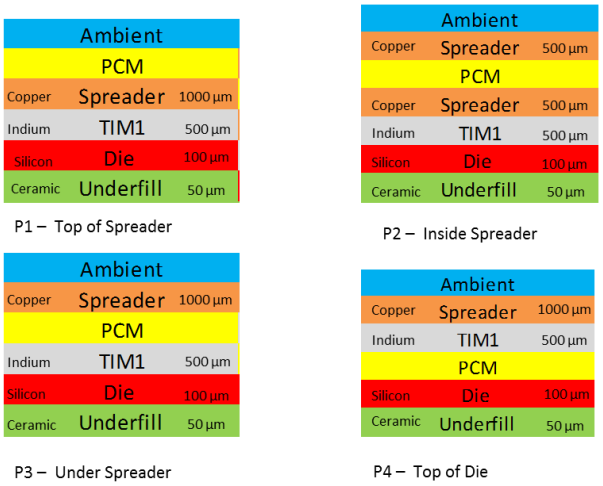
Οι διατάξεις αυτές, διαφέρουν ως προς τη θέση του στρώματος του υλικού αλλαγής φάσης, και το σημείο τήξης του. Κάθε ξεχωριστή διάταξη, προσομοιώθηκε για κάθε ένα από τα προηγούμενα προγράμματα και για στρώματα πάχους 100 και 200μm. Ο σκοπός αυτών των προσομοιώσεων είναι μια προσπάθεια να καθορίσουμε ποιες διατάξεις έχουν καλύτερη απόδοση και για ποιο σημείο τήξης.

Κάθε πρόγραμμα έτρεξε όπως προηγουμένως με 4 νήματα και στη συχνότητα των 2926 MHz. Αξίζει να σημειώσουμε ότι η αυτή η συχνότητα λειτουργίας καταλήγει σε αρκετά μεγαλύτερες καταναλώσεις ισχύος από αυτές που μπορεί να συντηρήσει το ολοκληρωμένο

απεριόριστα. Το αποτέλεσμα είναι ότι ο ρυθμός παραγωγής θερμότητας είναι ιδιαίτερα αυξημένος.

Υπό αυτές τις συνθήκες λοιπόν, προσομοιώσαμε κάθε διάταξη μέχρι να σημειωθεί η πρώτη παραβίαση των θερμικών ορίων λειτουργίας του ολοκληρωμένου. Το όριο αυτό τέθηκε στους 370 K, δηλαδή, περίπου 97 βαθμούς Κελσίου. Το ίδιο κάναμε και για μία διάταξη που δεν έχει καθόλου υλικό αλλαγής φάσης, την οποία χρησιμοποιήσαμε ως αναφορά. Ως απόδοση κάθε διάταξης, θεωρούμε τον χρόνο υπολογισμού που κερδίζουμε ως αποτέλεσμα του υλικού αλλαγής φάση, σε σύγκριση με την διάταξη αναφοράς.

Οι διατάξεις που διερευνήσαμε χρησιμοποίησαν υλικά αλλαγής φάσης με σημεία τήξης από 40 μέχρι 90 βαθμούς Κελσίου και βήμα 5 βαθμούς μεταξύ κάθε περίπτωσης. Όσον αφορά το σημείο τοποθέτησης του στρώματος του υλικού, δημιουργήσαμε 4 διαφορετικούς συνδυασμούς που ονομάσαμε P1-P4 οι οποίοι φαίνονται στο Σχήμα 8.



Σχήμα 8: Διατάξεις που χρησιμοποιήθηκαν για να προσδιοριστεί η βέλτιστη τοποθέτηση του υλικού αλλαγής φάσης

Τα στοιχεία που συγκεντρώσαμε από τις προσομοιώσεις που περιγράψαμε, φαίνονται στον Πίνακα 2. Οι χρόνοι που αναφέρονται είναι σε ms και αποτελούν το παραπάνω χρονικό διάστημα υπολογισμού που κερδίζουμε λόγω του υλικού αλλαγής φάσης. Αρνητικές τιμές σε αυτόν τον πίνακα δηλώνουν ότι η διάταξη έχει χειρότερη θερμική συμπεριφορά από τη διάταξη αναφοράς. Για λόγους σαφήνειας, το σημείο τήξης που παρουσίασε το μεγαλύτερο όφελος τονίζεται με πράσινο χρώμα, ενώ, η διάταξη που παρουσίασε το μεγαλύτερο όφελος σε σχέση με την τοποθέτηση του υλικού, τονίζεται με κόκκινο χρώμα.

Blackscholes	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	135	131	133	136	138	131	102	79	17	17	17
		P2	137	132	131	131	133	125	97	81	12	12	12
		P3	111	114	114	116	117	105	78	61	2	-1	-1
		P4	105	108	113	114	113	98	74	63	44	27	31
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	243	251	259	269	272	251	201	122	34	34	34
		P2	238	246	251	259	261	241	187	142	24	24	24
		P3	208	214	223	229	227	185	146	102	3	-2	-2
		P4	203	211	222	223	214	171	139	109	81	53	31
Bodytrack	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	92	92	92	92	99	92	90	52	19	19	19
		P2	98	97	92	92	92	92	90	69	13	13	13
		P3	92	92	92	98	92	92	92	69	1	1	1
		P4	92	92	98	92	92	92	92	97	59	28	35
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	286	307	313	326	341	106	92	66	49	49	49
		P2	309	309	312	318	331	106	92	88	43	43	43
		P3	109	110	272	110	108	102	92	89	2	2	2
		P4	108	110	284	112	108	103	100	97	75	64	56
Streamcluster	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	88	91	91	92	91	86	82	9	7	7	7
		P2	90	91	91	90	88	83	81	11	5	5	5
		P3	83	83	83	83	83	81	79	21	1	1	1
		P4	83	83	83	83	82	81	81	35	32	21	32
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	163	170	171	171	163	155	144	16	15	15	15
		P2	162	162	166	166	157	148	141	16	10	10	10
		P3	145	147	151	149	141	119	101	22	1	1	1
		P4	145	146	148	144	137	115	104	87	81	62	35

Πίνακας 2: Χρόνος (ms) που κερδίζεται μέχρι την πρώτη παραβίαση θερμοκρασιακών ορίων

Σύμφωνα με τα στοιχεία του Πίνακα 2, η διάταξη P1 έχει τις καλύτερες επιδόσεις σε όλες τις περιπτώσεις. Επιπροσθέτως, τα σημεία τήξης που βρίσκονται στο εύρος 50 με 60 βαθμούς Κελσίου, παρουσιάζουν τα μεγαλύτερα οφέλη ανεξάρτητα από την τοποθέτηση και το πρόγραμμα. Ωστόσο, θεωρούμε ότι το σημείο τήξης των 60 βαθμών είναι το επικρατέστερο, αφενός διότι παρουσιάζει τις καλύτερες τιμές στην πλειονότητα των περιπτώσεων, και αφετέρου διότι στις περιπτώσεις που κάποιο σημείο τήξης έχει μεγαλύτερη απόδοση, η διαφορά είναι πολύ μικρή. Παράλληλα, στη γενική περίπτωση θέλουμε να χρησιμοποιούμε όσο δυνατόν μεγαλύτερο σημείο τήξης για να αποφύγουμε το λιώσιμο του υλικού σε άλλες εργασίες που δεν έχουν σχέση με sprinting, και να σπαταλήσουμε έτσι την ικανότητα του.

Επιπροσθέτως, σχετικά με την τοποθέτηση του υλικού, το γεγονός ότι η διάταξη P1 παρουσίασε τις καλύτερες αποδόσεις είναι ενθαρρυντικό για δύο λόγους. Ο πρώτος, είναι το γεγονός ότι από όλες τις διατάξεις που χρησιμοποιήσαμε, η συγκεκριμένη είναι η πιο εύκολα υλοποιήσιμη και έχει ήδη δοκιμαστεί σε έρευνες. Ο δεύτερος, είναι το γεγονός ότι η διάταξη αυτή τοποθετεί το υλικό αλλαγής φάσης όσο πιο μακριά είναι δυνατό από τα ενεργά στοιχεία του

ολοκληρωμένου. Αυτό έχει σαν αποτέλεσμα χαμηλότερες θερμοκρασίες και μεγαλύτερη θερμοκρασιακή απόσταση μέχρι το σημείο τήξης. Η απόσταση αυτή, παρέχει ένα βαθμό ασφάλειας ότι δεν θα σπαταληθεί η αλλαγή φάσης σε μη κρίσιμες εργασίες.

Συνδυάζοντας όλα τα προηγούμενα, συμπεραίνουμε από τα πειραματικά δεδομένα και μερικούς ακόμα παράγοντες που χαρακτηρίζουν τα συστήματα που μελετάμε, ότι η διάταξη P1, με το υλικό αλλαγής φάσης να τοποθετείται πάνω από το χάλκινο κάλυμμα του ολοκληρωμένου και σε άμεση επαφή με το περιβάλλον, σε συνδυασμό με σημείο τήξης 60 βαθμών κελσίου αποτελεί τη βέλτιστη επιλογή.

Στη συνέχεια, θέλοντας να δούμε την επίδραση της αύξησης της ποσότητας του υλικού στη διάταξη μας, πραγματοποιήσαμε μια νέα σειρά προσομοιώσεων, χρησιμοποιώντας τη βέλτιστη επιλογή που αναφέραμε προηγουμένως και μεταβάλλοντας το πάχος του στρώματος του υλικού από 100μm μέχρι 700μm, αυξάνοντας κάθε φορά κατά 100μm. Οι προσομοιώσεις αυτές διεξήχθησαν πάλι μέχρι να σημειωθεί η πρώτη παραβίαση των επιτρεπτών θερμικών ορίων και για κάθε ένα από τα τρία προγράμματα που έχουμε χρησιμοποιήσει μέχρι τώρα.

Τα αποτελέσματα φαίνονται στον Πίνακα 3. Πιο συγκεκριμένα, αναγράφεται ο συνολικός χρόνος προσομοίωσης για κάθε περίπτωση σε ms. Επιπροσθέτως, στον πίνακα παρουσιάζονται άλλα δύο μεγέθη. Το ένα είναι η ποσοστιαία αύξηση της θερμικής αντίστασης από τα ενεργά στοιχεία του ολοκληρωμένου προς το περιβάλλον, ως αποτέλεσμα της προσθήκης περαιτέρω υλικού αλλαγής φάσης. Το δεύτερο είναι η ποσοστιαία αύξηση του χρόνου προσομοίωσης σε σχέση με την περίπτωση που δεν έχουμε καθόλου υλικό. Εμφανώς, το δεύτερο μέγεθος δεν προσφέρει κάποια περαιτέρω πληροφορία από τους συνολικούς χρόνους προσομοίωσης. Ωστόσο, το συμπεριλάβαμε σαν αναφορά για σύγκριση με την ποσοστιαία αύξηση της θερμικής αντίστασης.

Αξίζει να τονιστεί σε αυτό το σημείο, ότι οι χρόνοι προσομοίωσης, σχετίζονται άμεσα με το είδος της εργασίας που εκτελείται, τον αριθμό νημάτων που είναι ενεργά και τη συχνότητα λειτουργίας των πυρήνων, ενώ, η μεταβολή της θερμικής αντίστασης είναι καθορισμένη για συγκεκριμένο υλικό αλλαγής φάσης.

PCM Thickness	0	100	200	300	400	500	600	700	μm
Blackscholes	580	718	852	984	1115	1242	1368	1491	s
Bodytrack	626	725	967	1000	1201	1208	1519	1529	s
Streamcluster	380	471	543	629	713	796	901	985	s
Thermal Resistance increase relative to baseline									
	0	15.65	31.30	46.95	62.61	78.26	93.91	109.56	%
Simulation Time increase relative to baseline									
Blackscholes	0	23.79	46.90	69.66	92.24	114.14	135.86	157.07	%
Bodytrack	0	15.81	54.47	59.74	91.85	92.97	142.65	144.25	%
Streamcluster	0	23.95	42.89	65.53	87.63	109.47	137.11	159.21	%

Πίνακας 3 : Συνολικός χρόνος προσομοίωσης μέχρι την πρώτη παραβίαση επιτρεπτών θερμικών ορίων

Με αυτό σαν δεδομένο, παρόλο που για όλα τα προγράμματα που χρησιμοποιήσαμε, τα οποία διαφέρουν σημαντικά στη λειτουργία τους, ο χρόνος που κερδίσαμε είναι μεγαλύτερος ποσοστιαία από την αύξηση της θερμικής αντίστασης, πρέπει να λάβουμε υπόψη μας και άλλους παράγοντες.

Ο πιο σημαντικός από αυτούς, είναι η επίδραση της πρόσθετης θερμικής αντίστασης στο σύστημα μας. Στην περίπτωση μας, υποθέσαμε ότι το σύστημα βρισκόταν σε αδράνεια πριν ξεκινήσει την εργασία που του αναθέσαμε. Για το λόγο αυτό η αρχική θερμοκρασία των στοιχείων του ολοκληρωμένου τέθηκε στους 300K. Όταν αυξάνεται όμως η θερμική αντίσταση του συστήματος, αυξάνεται πρώτιστα και η διαφορά θερμοκρασίας μεταξύ πυρήνα και περιβάλλοντος. Η αύξηση αυτή, επειδή υπάρχει γραμμική σχέση μεταξύ της κατανάλωσης ισχύος, της θερμικής αντίστασης και της διαφοράς θερμοκρασίας, γίνεται ακόμα πιο έντονη σε έντονες, υπολογιστικά, συνθήκες.

Για παράδειγμα, ας υποθέσουμε μια αυθαίρετη, σταθερή κατανάλωση ισχύος που ανεβάζει τη θερμοκρασία του πυρήνα στους 40 °C. Αυτό σημαίνει ότι έχουμε διαφορά 15 βαθμών μεταξύ περιβάλλοντος και πυρήνα. Χρησιμοποιώντας υλικό αλλαγής φάσης πάχους 700μm, η θερμική αντίσταση του συστήματος περίπου διπλασιάζεται. Αυτό σημαίνει ότι η νέα διαφορά θερμοκρασίας θα είναι 30 βαθμούς και η νέα θερμοκρασία του πυρήνα, 55 °C. Λαμβάνοντας υπόψη ότι το υλικό μας λιώνει στους 60 °C, και παρά το γεγονός ότι βρίσκεται μακριά από τον πυρήνα, όπου οι θερμοκρασίες είναι χαμηλότερες, βλέπουμε ότι ανάλογα με την προηγούμενη κατάσταση του συστήματος, τα οφέλη σε απόδοση μπορεί να είναι πολύ μικρότερα και να μην δικαιολογούν το παραπανίσιο υλικό

Σε αντιδιαστολή, σε συνθήκες ηρεμίας, ο πυρήνας είναι μόνο 2 -3 βαθμούς πιο θερμός από το περιβάλλον. Διπλασιάζοντας τη θερμική αντίσταση του συστήματος θα έχει ως αποτέλεσμα ο πυρήνας να είναι 4 – 6 βαθμούς θερμότερος. Η διαφορά σε αυτή την περίπτωση είναι πολύ πιο ανεπαίσθητη.

Από μια διαφορετική οπτική γωνία, η αύξηση της θερμικής αντίστασης αυξάνει κατά πολύ τη σταθερά χρόνου του συστήματος. Αυτό συμβαίνει διότι η σταθερά χρόνου είναι συνάρτηση της θερμικής αντίστασης αλλά και της θερμικής χωρητικότητας που επίσης αυξάνεται προσθέτοντας υλικό. Η αύξηση της σταθεράς χρόνου, συνεπάγεται ότι το σύστημα κάνει περισσότερο χρόνο να ανεβάσει θερμοκρασία αλλά ταυτόχρονα χρειάζεται και περισσότερο χρόνο για να την αποβάλλει.

Σύμφωνα με τα προηγούμενα, γίνεται φανερό ότι προσθέτοντας υλικό αλλαγής φάσης, μεταβάλλεται το θερμικό προφίλ του ολοκληρωμένου. Παρόλο που η δήλωση αυτή είναι προφανής, αυτό που θέλουμε να τονίσουμε είναι ότι η επιλογή του πάχους του υλικού αλλαγής φάσης, πρέπει να γίνει σε συνάρτηση με το είδος των εργασιών που θέλουμε να εκτελεί συνήθως το σύστημα. Για παράδειγμα, συστήματα που συνήθως εκτελούν σύντομες εργασίες οι οποίες εμφανίζονται συχνά, θα ήταν βέλτιστο να χρησιμοποιούν λίγο υλικό. Αντίθετα, συστήματα τα οποία εκτελούν πιο μεγάλες εργασίες οι οποίες εμφανίζονται πιο αραιά, θα είχαν μεγαλύτερο όφελος από μεγάλες ποσότητες υλικού παρά την αύξηση στη θερμική αντίσταση.

Acknowledgements

The current thesis is the result of my work in collaboration with the Microprocessors and Digital Systems Laboratory (MicroLab) of NTUA. I would like to thank my supervisor, Prof. Dimitrios Soudris for the trust he showed in me in conducting this thesis. His direct and thorough explanations, early on, of the challenges involved, gave me a unique perspective and were most appreciated. In addition, his constant presence and overall conduct was both encouraging and helpful at all times. I would also like to thank Postdoctoral Researcher Sotiris Xydis for his support and patience. This work would not have been completed without his contribution and guidance. In addition, throughout all the obstacles that occurred, his suggestions were always most insightful and thorough. Lastly, I would like to thank all the members of the laboratory with whom I have interacted during the course of this thesis. Their professionalism coupled with their friendly manner, enabled me to study in an environment both inspiring and pleasant.

List of Figures

List of Tables

CHAPTER 1

1. Introduction

A great many years have passed since the publication of the article [1] of Gordon E. Moore that led to what we now call Moore's law, that is, the doubling of transistor count on chip every 18 months. Moore's law, in essence, predicted a shrinking trend in transistor size leading to a corresponding growth in transistor count per unit area. While not as commonly discussed, a major factor that allowed this scaling to hold true, up to a certain point, is what we refer to as the Dennard Scaling model.

Robert H. Dennard, after whom it is named, observed that voltage and current should be proportional to the linear dimensions of a transistor. Thus, as transistors shrank, so did the necessary voltage and current, meaning that power is proportional to the area of the transistor. Dennard's scaling theory showed how to reduce the dimensions and electrical characteristics of a transistor proportionally to enable successive shrinks that simultaneously improved density, speed and energy efficiency.

According to Dennard's theory [2], with a scaling ratio of $1/\sqrt{2}$, the transistor count doubles (Moore's Law), frequency increases by 40%, and the total chip power stays the same from one generation of process technology to the next, on a fixed chip area.

While approximately accurate, this model does not account for the increased leakage currents dominant in small sizes. In leakage dominated, deep sub-micron technology nodes, further reducing threshold voltage results in an exponential increase in leakage power. Hence, threshold voltage is no longer scaling, and, as a consequence, supply voltage cannot be scaled further without impacting performance. Thus, although we can still pack more transistors per area with technology scaling, the switching power per transistor is not scaling commensurately, and hence power density has been trending upwards. Coupled with the physical limits imposed by device packaging and cooling technology on the peak power and peak power density, this results in the so-called Dark Silicon era. The term "dark silicon" derives from the fact that as chip power density increases, more and more chip area must remain unpowered (dark).

This new constraint imposed by dark silicon means that not all the transistors on the chip can be simultaneously powered on at full performance for a given thermal design power (TDP). The TDP is the maximum amount of power that can be supplied to the chip to ensure that the chip will operate within the safe range, meaning, below thermally safe temperatures. If the TDP is violated, the chip will generate heat at a faster pace than can be dissipated by the cooling system, which will eventually lead to overheating of the chip components [3].

With the end of Dennard scaling, as previously explained, process technology scaling can sustain doubling the transistor count in every generation, but with significantly less improvement in switching speed and energy efficiency [4]. In other words, the transistor count could continue to increase under a specific frequency limit, at the cost of increasing power density. As a result, the microprocessor industry has shifted to multicore scaling, in order to exploit the still increasing transistor numbers. Multicore scaling, translates to increasing the number of cores per die at each generation instead of focusing on creating a single, faster core. To further elaborate, the aim is to utilize more, energy-efficient, cooler-running processing cores instead of one, increasingly

powerful and increasingly consuming. Multicore chips are not intended to run as fast as single core models, but improve overall performance by handling more work in parallel [5].

The main advantage of multicore systems is that raw performance increase can come from increasing the number of cores rather than frequency, which translates into a slower growth in power consumption. However, this approach is not ideal because it requires tasks that can be divided equally among cores, in order to reap the most out of the potential performance gain, which, quite frequently, is not the case [6]. In addition, with the failure of Dennard Scaling and thus, voltage-scaling, even core count increases are not without limits. As previously stated, core scaling results into a slower growth of power, but still an upward trend. Consequently, it stands to reason that even this scaling will eventually come to an end. Hence, the leap to multicore is not based on a breakthrough in programming or architecture and is actually a retreat from the more difficult task of building power-efficient, high-clock-rate, single-core chips [7].

Given the abundance of transistors in dark silicon chips, the question ultimately becomes *if* and *how* they can be harnessed to improve performance within a power or peak temperature constraint. Much of the existing work in literature addresses this question, based on different design philosophies that include the use of architectural heterogeneity, specialized cores, approximate computing and devices that employ near-threshold voltages to enable a larger fraction of the chip to be powered on, albeit at lower voltage level, coupled with other power management schemes for dark silicon.

Regarding architectural heterogeneity and specialized cores, research work has focused on exploiting the dark silicon area for designing specialized cores, incorporating heterogeneity, and/or application-specific, hardware accelerators. In essence, this approach focuses on using the surplus in silicon area to provide better suited hardware in terms of energy and performance that can be used in an on demand basis, depending on the task at hand.

Approximate computing relies on trading energy efficiency with accuracy, especially for error-tolerant applications like vision, machine learning etc. Approximate computing techniques at various levels of design abstraction have been discussed, ranging from circuit level techniques, to approximate data paths and programming language support.

Near threshold computing, presents an approach to utilize dark cores by turning on a larger fraction of the chip but at voltages close to the threshold voltage. This approach works well for applications with high thread-level parallelism, but also exhibits high sensitivity to process variation and power supply fluctuations.

In terms of power management, recent research aims at run-time mechanisms to efficiently utilize the thermal design power (TDP) budget in order to maximize performance of cores that can be homogeneous, micro-architecturally heterogeneous or homogeneous but synthesized with different power/performance targets.

A technique called computational sprinting leverages dark silicon to power-on many extra cores for a very short time period to facilitate sub-second bursts of parallel computation. During this time window, the active cores consume power that significantly exceeds the sustainable TDP budget, but the cores are immediately power-gated after the sprint allowing the chip to cool down.

Alternate methods are Intel's Turbo Boost and AMD's Turbo CORE technologies that leverage the temperature headroom to favor high-ILP applications by increasing the voltage/frequency of a core while power-gating others [3].

In an effort to assist computational sprinting, the use of phase changing materials (PCMs) has been investigated as a passive cooling technique. PCMs are compounds that store large amounts of latent heat during phase change from solid to liquid. PCMs absorb this heat at a near-constant temperature and hence act like large thermal capacitors. These properties have led to the use of PCMs in cooperation with computational sprinting aiming to extend the sustainable sprinting duration [8].

In this diploma thesis, the aim is to address the performance bottleneck imposed by the multicore era and the ever increasing percentage of dark silicon. With regard to this goal, we consider computational sprinting combined with the use of phase change materials to be quite promising. Therefore, we sought to showcase the impact of different PCM configurations and attempt to find the best suited properties regarding placement, material thickness and melting temperature.

The goal of this analysis is to identify setups augmented with phase change materials that allow even better results to be achieved through the use of sprinting techniques. To this end, a simulation framework has been formed, consisting of four basic components: a hardware simulator, a power modeling tool, a thermal simulation tool and a collection of python scripts unifying each separate component into a single entity. Although this simulation framework was used to simulate a specific chip enhanced with PCM under a specific set of workloads, its capabilities far exceed the implementation we used. This will be highlighted when the framework is presented in detail.

In order to properly simulate the behavior of PCMs, the thermal modelling tool has been modified. It has been altered to be able to implement a model proposed in [9], named *apparent heat capacity method.* This method simulates the behavior of phase change materials by assigning a nonlinear, temperature dependent, specific heat capacity to the PCM layer. The benefit of this approach is the ability to simulate non-uniform PCM melting and, by way of the specific implementation, materials of different physical properties. In addition, via the native flexibility over the placement of layers offered by the thermal simulator, comes the potential to experiment with the placement of PCMs in the chip stack.

Subsequently, using the derived framework, this thesis goes on to show that traditional models used for the components of the chip package are not suitable for simulations involving PCM-enabled sprinting chips. This results from the fact that said models contain inaccuracies whose magnitude is an increasing function of core frequency. Furthermore, these models fail to properly simulate the uneven heat distribution that results from heterogeneous power consumption of various components (be it across components of the same core or across different cores). This fact is propagated across all layers of the chip and results in completely different heat distribution profiles. Accurate heat distribution modelling is considered of the utmost importance since we aim

to simulate uneven PCM melting. Thus, a more accurate model is introduced, taking into account the increasingly weighing factor of thermal interface materials.

Using the constructed framework that includes the enhanced thermal model and the accurate model of the chip package stack, we attempt to determine the best phase change material configuration for a given chip, under a number of different workloads. The variables addressed are the PCM melting point and thickness, as well as the placement of the PCM layer in the chip stack.

Lastly, on account of the observation that components that consume more power result in concentrated regions of PCM that melt (that is, exhaust their heat capacity) faster, we experiment with heterogeneous PCM layers. These layers consist of regions with different melting points, such that the effect of more active regions in the silicon is reduced by enhancing lateral heat spreading across other, cooler areas of the PCM layer.

In retrospect, although this thesis aims at more efficient ways to utilize the resources available in recent multicore platforms, it also contributes a very adaptable and complete simulation framework for PCM-enabled multicore chips, as well as insight into phase change material configurations and their corresponding impact.

CHAPTER 2

2.1 Theoretical Background

2.1.1 Computational Sprinting

With the apparent end of Dennard Scaling and the shift of the microprocessor industry to multicore designs, more and more research is oriented towards finding ways to better utilize the resources available in many core chips. One of the most promising, and popular approaches to this end, is that of computational sprinting. While this technique has already been presented briefly, a more detailed description is in order.

Computational sprinting, or sprinting, as the term will be used subsequently for the sake of brevity, activates reserve cores (parallel sprinting) and/or boosts frequency and voltage (frequency sprinting) for bursts of intense computation, to power levels that exceed the system's sustained cooling capabilities by an order of magnitude or more. During sprinting, chip temperature does not spike instantaneously, although the chip generates heat faster than the system dissipates it. Instead, the system absorbs heat by virtue of its inherent thermal capacitance, that is, the property that materials can buffer significant heat as they rise in temperature. This property causes temperature to rise over an extended, albeit still short, time interval. When it reaches a threshold value, sprinting terminates, and restraining actions are being enforced in order to complete the remainder of the computation in progress, at lower power levels, while heat buffered to the chip materials is dissipated to the ambient [10], [11].

While the idea seems to be quite simple, many factors contribute to the existence of a variety of different sprinting implementations. First and foremost, any sprinting methodology is designed with regard to a specific platform type. The most usual case are chips intended for mobile devices without, however, precluding the existence of implementations for chips that are used in servers or even personal computers. As previously mentioned, Intel and AMD turbo technologies are examples of limited forms of frequency sprinting, widely used in multicore chips, mobile or desktop oriented. With that example in mind, it is worth restating at this point, that sprinting approaches may involve frequency sprinting, parallel sprinting, or both, depending on the case.

As expected, depending on the platform, a number of consequent elements must be taken into account, thus differentiating each sprinting implementation. Despite the fact that many of these elements are interdependent, and not necessarily in a direct manner, we will attempt to showcase some of the most important, disregarding the correlations between them. The objective is to give a general sense of the huge number of disparate approaches, the complexity of designing a sprinting scheme, and a brief theoretical background that we will subsequently use to analyze related research.

One way to differentiate between sprinting policies is with regard to the applications they are aimed towards. In some cases, only a specific set of workloads is intended to employ sprinting as a consequence of favorable intrinsic characteristics. For example, as described in [12], applications that demonstrate short bursts of intense computation, punctuated by long idle periods, are suitable candidates for sprinting. Some applications that fit this pattern are image processing and computational photography tasks (such as panoramic stitching and image noise reduction),

navigation route planning, and natural language processing (speech recognition and translation). In other cases, no special consideration regarding the applications suitable for sprinting is performed, and the system sprints if available tasks and resources are both present. These systems usually perform a duty cycle type of operation, defining a time cycle, sprinting for a fraction of that cycle and switching to sustained operation for the remainder. Lastly, special cases perform a dynamic evaluation and decide whether it is efficient for an application to sprint, based on performance characteristics.

In any case of sprinting, a control mechanism is enforced, to ensure that chip temperature levels remain within certain bounds. We distinguish between two basic control types, reactive and predictive. Reactive control models take necessary restraining action when a violation occurs. In contrast, predictive models, use metrics to evaluate whether a thermal violation is about to occur in the immediate future and react accordingly. Predictive control depends greatly on the accuracy of the employed model. In the general case, predictive control types ensure less thermal strain on the chip and are often coupled with reactive policies in case of a failure to predict a temperature violation. On the other hand, they carry a certain computational overhead. This is a direct consequence of the fact that predictive models perform a series of computations in order to simulate, albeit in a simplified manner, the behavior of the system in the future. Depending on the implementation, the detail, and the range of variables associated with each predictive model, such computational overhead may be substantial. Reactive policies almost always are enabled by a temperature limit violation.

Despite the fact that control mechanisms, when and if enabled, override any resource allocation scheme, they are a separate entity. Resource allocation schemes or allocation policies, as denoted by their name, decide the optimal allocation of chip resources in an attempt to maximize performance and/or energy efficiency. This optimization, might be referring to an application, a thread, a system average or some other module. While a resource allocation scheme might also be limiting the distribution of resources, it does so in an effort to achieve the best response out of the system. In contrast, control mechanisms throttle the function of components, to prevent damages due to non-nominal operation.

Another important aspect of a sprinting policy, closely related to the control mechanism, is the overheat policy. By overheat policy we mean the series of actions that take place when a violation, overheating of a component, occurs. According to the information presented above, the system would revert to a sustained operation mode, in order to cool down, however, a certain variability in such modes can be observed. The most common case of overheat policy, is to shut down all active cores except one, pack all remaining active threads and continue computation at the nominal frequency until significant thermal headroom is recovered. Other, more sophisticated strategies, choose to power down only those cores that report a violation, and migrate or pack their threads depending on the existence of free cores to accommodate them. A different approach is to define a number of sustainable combinations of active cores and frequency/voltage levels and choose the most appropriate, depending on the workload.

Lastly, a distinguishing factor between sprinting approaches is the homogeneity, or not, of the multicore chip. Most approaches in the literature address homogeneous platforms in spite of the fact that combining heterogeneity with computational sprinting is generally regarded to be a promising research area.

The key points from this section are summarized in Figure 2.1. Although it was mentioned in the beginning of this section, let us repeat that what we presented is only a simplistic outline of a countless number of considerations related to a sprinting methodology. For example, an allocation policy is not wholly characterized by the applications it is intended for. A lot of other factors may be taken into account in order for the allocator to decide when, where, for how long, at what intensity and so on. From a higher level of perspective, the factors presented in Figure 2.1 are not the only ones distinguishing one methodology from another.

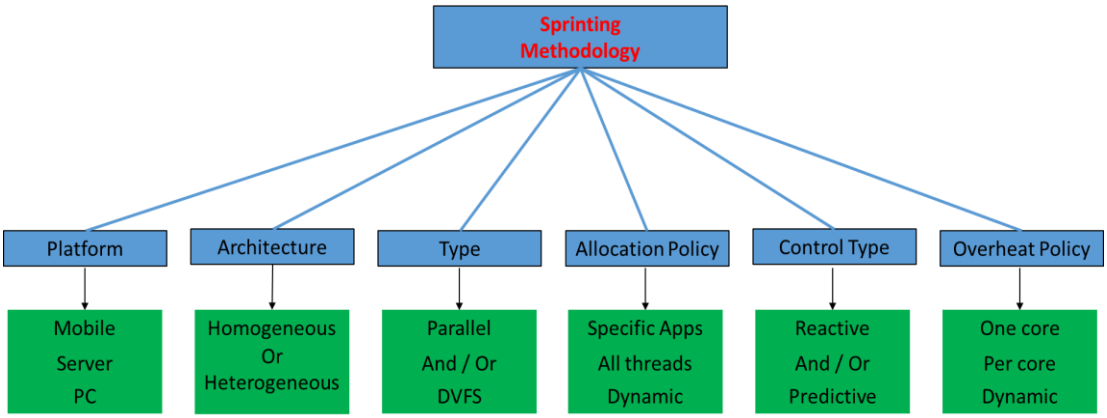


Figure 2.1: Basic characterizing factors of sprinting methodologies

2.1.2 Phase Change Materials (PCMs)

Phase change materials, in general, are compounds that are able to store large amounts of thermal energy during phase change. In the most common case, the phase change from solid to liquid, and vice versa, is exploited. From this point of view, PCMs are substances with high heat of fusion. Heat of fusion or latent heat of fusion, is the energy required to transform a certain mass of a material from solid to liquid. The term latent, accounts for the fact that phase change occurs at near steady temperature, a metric for the internal energy of a substance, hence, the material absorbs heat that is latent, meaning hidden. Heat absorbed by a material that results in an observable temperature difference is termed sensible heat.

In computational sprinting, the topic of our focus, such materials are exploited by virtue of their ability to store large amounts of heat at near constant temperature. This results in a sustained considerable heat flow towards the PCM, due to the fact that heat flow is linearly dependent on the temperature difference between surfaces. The ultimate result of this fact, is that the use of PCMs in sprinting systems leads to smoother temperature profiles, that is, longer sprinting durations.

As expected, the latent heat of fusion of a material, or more accurately, the specific heat of fusion, is a weighting factor of both its suitability for and impact to, sprinting approaches. The specific heat of fusion is the energy required for phase change, from solid to liquid, per unit mass. Obviously, the specific heat of fusion is a material property and independent of size or extent of a sample [13].

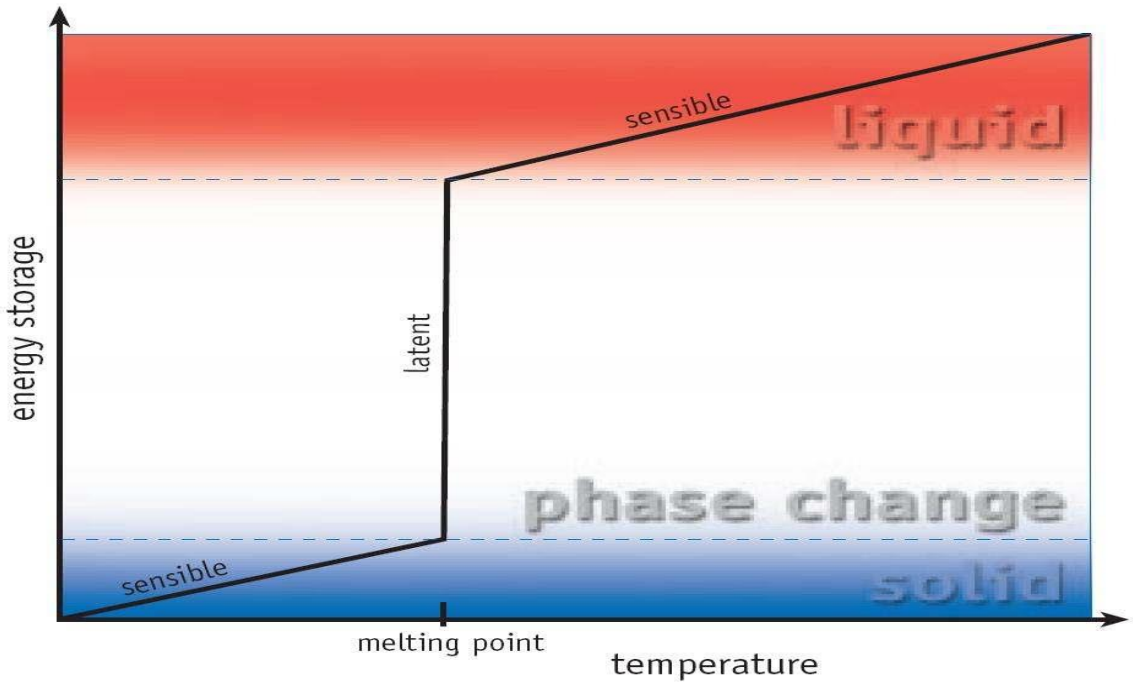


Figure 2.2: PCM behavior and latent heat storage

Another important characteristic for a PCM, is thermal conductivity. Thermal conductivity is the property of a material to conduct heat. It represents the rate of heat transfer. Heat transfer occurs at a lower rate across materials of low thermal conductivity than across materials of high thermal conductivity. Correspondingly, materials with high thermal conductivity like copper, are used in heat sinks, while materials of low thermal conductivity are used as thermal insulation [14]. For PCMs of our interest, high thermal conductivity is paramount, as is the case with all materials involved in chip packaging. More to the point, uniform heat distribution, an immediate result of high thermal conductivity, given enough time, is essential in order to exploit the maximum out of the heat storage that a PCM offers. Typical thermal conductivity values of PCMs indicate at least an order of magnitude difference to that of heat spreader and heat sink materials. Consequently, the placement in the chip stack and size of PCM layers warrant a careful investigation for their use to be productive.

Furthermore, careful consideration must also be given to the PCM melting temperature. While this term is self-explanatory, the ideal value for a sprinting system is open for debate. Usually, a melting point close to the critical temperature for chip operation is considered. To elaborate, a slightly lower melting point than the critical temperature is regarded as optimal, to

account for a degree of latency in the heating of the PCM layer. Generally, PCMs with higher melting points are considered useless at best, for obvious reasons. In addition, PCMs with quite low melting points are also considered non-ideal since their advantage of storing large amounts of heat is wasted in non-critical temperatures and their existence in the chip stack hampers heat conduction. However, since these remarks are quite vague in nature and, to our knowledge, no extensive research regarding this factor has been conducted, a series of tests will be later presented, aiming to find the best PCM melting point for our test system.

In summary, PCMs are used in systems with computational sprinting in mind because they have been proved to be able to extend the maximum sprinting duration. The key elements of a phase change material are its specific heat of fusion, thermal conductivity and melting point. It is beneficiary for thermal conductivity and specific heat of fusion to be as high as possible. In contrast, regarding the melting point only vague outlines and isolated examples are available. For this reason, an exploration has been conducted and will be presented in detail.

2.2 Related Work

2.2.1 Computational Sprinting on a Hardware/Software Testbed [10]

In this paper parallel and frequency sprinting is demonstrated for a configuration imitating a mobile chip. To this end, a desktop four core system is modified, by removing the heat sink and adding a variable speed fan, so as to be able to sustain the indefinite operation of only one core at the lowest user-selectable frequency. All other modes of operation normally available to the chip, are considered sprinting modes for this configuration. In order to show various aspects of sprinting, a set of parallel workloads is used.

At first, the aim is to show the benefits of sprinting when a computation can be completed at the maximum parallel and frequency levels without exhausting the system's thermal capacitance. The resource allocation scheme and application work size are configured accordingly. An average speedup of 6.3 is reported. Consequently, the energy impacts of sprinting are examined. To do so, the same set of applications is considered, at all possible sprinting combinations. Counterintuitively, the authors demonstrate that when sprint intensity is selected appropriately, sprinting can improve energy efficiency as well as responsiveness.

Bigger work sizes are also examined, at maximum sprinting intensity, and a reactive control with regard to temperature, measured at a fixed time interval with on-die sensors, is enforced. The overheat policy of this configuration is to revert to sustained one-core execution at the lowest frequency, pinning all threads to this single core and disabling all others. This type of operation results to performance and energy penalties, more prominent depending on the application, that increase as the workload increases. In order to mitigate the oversubscription penalty resulting from pinning active threads to a single core, a methodology is presented that essentially allows to dynamically alter the thread count of an application to match that of active

cores. With the implementation of the proposed methodology, even the most penalized application adopts an almost neutral performance and energy profile for big workloads.

An allocation scheme, termed adaptive sprint pacing, is presented, intended to capture the benefit of sprinting for short computations, but at the same time extend the length of computations for which sprinting improves responsiveness. The idea outlined is to sprint at full intensity until half of the thermal capacity is consumed, and then switch to less intense and more power-efficient sprints by keeping all cores active but lowering their frequency levels. Indeed, this allocation scheme is seen to capture beneficial effects for a greater range of worksizes.

The use of PCM, specifically paraffin wax, is examined in order to increase sprinting duration. The response of the system at the max parallel but lowest frequency sprint mode is measured and compared against the configuration with no PCM at all and other cases, involving materials not equally suitable for augmenting the system's thermal capacitance. The best scenario, that of paraffin wax, resulted in a 6x increase in sprinting duration. In all scenarios, the extra layer was placed on top of the heat spreader, meaning, since no heat sink was present, between the spreader and ambient.

Lastly, a duty cycle type of execution is researched. This means defining a time period, sprinting for a fixed time length and switching back to sustained execution for the remainder and repeat indefinitely. Again, the sprinting mode selected is the maximum parallel and minimum frequency sprint. This sprint and rest execution was reported to provide both better performance and energy efficiency.

2.2.2 Safe Computational Re-Sprinting via Model Predictive Control [15]

In this work, a sprinting architecture for an embedded device is presented consisting of a 16 core chip. Both frequency and parallel sprinting is enforced, although, in an indirect manner. The authors assume fixed power consumption at maximum and minimum frequencies, in a worst case scenario basis. The same is true for the power consumed when idling. A full sprint corresponds to using all cores at maximum frequency and utilization, and rest mode to only one core operating at full capacity. The assumed chip is supposed to be augmented with a PCM-copper composite, interposed between the die and package.

It is worth noting that the thermal model adopted, assumes a uniform heat distribution across core area and across the PCM layer. With this configuration, the effectiveness of the PCM composite is highlighted by comparing the thermal performance of the chip when a copper heat spreader or a PCM without the copper enhancement are used instead the composite. As expected, the copper enhanced phase change material provided the larger sprinting duration.

Before introducing their proposed model, the authors define a special case of critical tasks that have a finite pre-specified duration and are issued at regular time intervals. These tasks are required to sprint at full power until completion. With that definition in mind, a two layer predictive controller is described.

The lower layer satisfies power levels requested by the higher layer when no violation of thermal bounds is predicted. In any other case, the maximum power that does not result in a thermal violation is assumed, if such a value is greater than the power corresponding to the lower operating frequency. To the event that this condition is not satisfied, idle power is enforced, essentially meaning that a core has been put to idle status.

The higher layer assigns maximum power levels to each core when the PCM internal energy is not predicted to exceed a certain value. This value is computed in order to ensure that enough thermal headroom will be available for a critical task to sprint at full capacity. If a violation is predicted, then the controller will assign the maximum power level acceptable in a similar manner as described previously, with the sole difference that the control variable in this case, is the PCM internal energy. Furthermore, it is worth noting that this higher layer controller can be disabled, when no critical tasks are considered.

The proposed model is validated against a threshold based approach, where each sprinting request is executed at maximum capacity until each core but the first is forced to shutdown. This comparison is performed with mixed workload scenarios for cases where critical tasks are assumed along with other generic ones, and cases where only generic tasks are issued. In the latter case, the approach presented by the authors clearly outperforms the threshold based. When critical tasks are considered, the proposed model ensures the required full sprint duration for critical tasks, but suffers from a small overall performance loss to do so.

To conclude the outlined approach, non-nominal conditions are assumed, that is, high ambient temperature and power consumption for each core, for a mixed criticality workload. Suffice it to say that the performance of the controller satisfies the expectations even at the specified challenging circumstances.

2.2.3 Thermal Management Using PCM – Based Heatsinks [16]

This paper investigates the interaction between PCM enhanced heatsinks and systems running specific benchmarks. An Intel Nehalem four core platform is simulated using the Sniper Simulator [17] coupled with McPAT [18], to produce power values. These power values are then fed to Hotspot to evaluate the thermal interaction between the chip and other components of the package, for different benchmark workloads.

While this work accounts for uneven distribution of heat across core areas, as a result of different utilization characteristics for each component, due to variability of the workload, the PCM layer is modeled as a singular block. This type of modelling, indirectly assumes uniform heat distribution and uniform melting of the PCM layer. As we will show later, this condition is almost never satisfied. Moreover, this single block modelling of the PCM layer, mitigates the benefits of simulating uneven heat distribution in the underlying chip area.

A small number of workloads is enough to demonstrate the theoretically expected behavior of the PCM heatsink, thus highlighting the benefit offered by the phase change material. For

reference purposes, the behavior of the PCM selected, Lauric Acid, is compared against that of water for the corresponding temperature ranges that ice melts.

While this paper does not enforce any sprinting methodologies, it is included because it contributed to the formulation of the overall framework that will be presented later. In addition, it is the only one that associates the use of phase change materials with uneven heat distribution in the core area despite the mitigating factor involved in modelling a PCM layer as a single block.

2.2.4 Modeling and Analysis of Phase Change Materials for Efficient Thermal Management [9]

This work proposes a thermal model to simulate the behavior of phase change materials that is able to account for non-uniform melting and heat distribution at the PCM layer. In order to achieve that, the apparent heat capacity method [19] is used. This method was also used in our thermal model and will be presented in detail in chapter 3. The PCM layer in this research is interposed between the die and the copper heat spreader.

This model is then validated against COMSOL and another, single block approach. The proposed approach is proven to be quite accurate and also more efficient than COMSOL. At the same time, the single block model is proven to be quite inconsistent, leading to big temperature errors and, of course, unable to capture spatial effects like non-uniform melting of the PCM.

Consequently, a 12 core system is simulated by a framework using the Gem5 simulator, McPAT and HotSpot thermal simulator. The objective is a design space exploration of PCM properties and their impact on the thermal performance of a chip. Results from experimenting with the PCM thermal conductance and thickness are presented. Overall, having the highest conductivity available is preferred in all cases while the thickness of the selected material is not an easy choice. In the general case, thick layers result in hotter thermal profiles but PCM-aware control mechanisms might leverage the extra thermal capacity to increase performance benefits.

Lastly, the authors demonstrate that using single block models for phase change materials results in inconsistencies that are exacerbated when workload distribution among cores is highly heterogeneous. Specifically, single block models, result in longer time before the layer starts melting when few cores are active, and, during melting, they exhibit a constant temperature until 100% of the PCM is melted. These erroneous results, lead to a number of over and under-estimations of core temperatures and consequently, throttling instances if a per-core thermal controller in enforced.

2.3 This Work

2.3.1 Objective

The aim of this thesis is to research the efficient use of phase change materials in systems that intend to employ computational sprinting. This goal was motivated by the fact that even though phase change materials have been presented, in recent research, to be able to increase sprinting duration for such systems, there has been no extensive study regarding their optimal characteristics and placement.

2.3.2 Key Differences

One of the major and most prominent differences in this work, lies within the thermal model adopted. First of all, in most research, the power consumption of a core is either assigned a steady value, reflecting a worst case workload scenario, or is simulated with suitable software and then aggregated to a single value. In either case, the resulting power is usually evenly distributed across the core area. Other more conservative approaches, might split the core simulated into a small number of logic blocks, containing multiple elements, aggregate the values of those elements, and then assign the computed power estimate to the logic block. While the latter case is more accurate, we considered it a coarse grained representation of the heterogeneous heat distribution across the core die. In our work, a fine grained floorplan, and corresponding power, is used with interesting results.

In addition, regarding the modeling of phase change materials, most research adopts computational methods to account for the phase change, and models the PCM as a single block. In our work, a more accurate method is used, the apparent heat capacity method, which is validated with good results against state of the art thermal simulators, coupled with a multi-node grid representation. While the model was adopted from [9] and used there, it was not implemented in that work in collaboration with a framework that accounts for the variable heat distribution along the area of a core, the first major difference mentioned earlier.

Regarding these two facts, our work accounts for non-uniform heat distribution at all possible levels. Across the die of one core, due to different component utilization, across the chip die, due to different usage of each core and across PCM and every other layer in the chip stack. This approach, as will be demonstrated consequently, will yield some results that deviate from what was anticipated.

To make matters worse, the effect of thermal interface materials (TIMs) is neglected in almost all the related work we examined. One might argue that since these materials are by design slim in the chip stack, a simplified model omitting these interconnecting layers will benefit in performance at the cost of a small accuracy loss. However, we will show that this is not the case, and in the platforms of our interest, characterized by big temperature gradients and time windows

of mere milliseconds, the impact of thermal interface materials is palpable. This impact is further magnified in modern high power chips because TIMs are the essential bottleneck in heat transfer between the die and any passive cooling solution, a bottleneck that becomes of increasing importance as heat density increases.

Naturally, research conducted in real hardware testbeds, does not suffer from any of the above limitations. Be that as it may, such research is limited in a particular setup and is unable to explore other hardware configurations. In addition, real hardware frameworks are unable to explore many diverse configurations with phase change materials or analyze them in a detailed manner. In addition, in this thesis, some PCM configurations are purely theoretical and are examined even though an actual chip with the specific characteristics might not be possible to be constructed. The aim was to showcase the most promising setups and perhaps steer research towards finding ways to implement them.

2.3.3 Contribution

The main contribution of this thesis is a robust and quite flexible framework that performs full system simulation, from hardware performance to thermal results. This framework functions in a cycle wise basis allowing for resource management decisions to be effected during run-time. Also, as was mentioned before, the thermal simulator integrated in the framework, is able to simulate the use of phase change materials anywhere in the chip stack. While the framework will be analyzed in great detail in chapter 3, it is worth noting that even non uniform PCM layers can be simulated. In addition, the effect of neglecting thermal interface materials in systems intending to employ sprinting is presented along with a proposed, more accurate, thermal model for the chip stack. This model achieves better accuracy regardless of the presence of phase change materials in the chip stack. Furthermore, an exploration regarding PCM thickness, melting point and placement has been conducted. The results are being presented in chapter 5 in an effort to provide insight on how to use such materials effectively. At the same time, a number of common simulation artifacts that compromise the results of already conducted research will be demonstrated. Lastly, the benefits of using heterogeneous PCM layers, a completely novel idea is analyzed in chapter 6.

CHAPTER 3

3.1 Overview

The framework constructed for the purpose of this thesis is oriented towards simulating the operation of a multicore chip to the greatest detail possible, while keeping simulation times to a reasonable value. It was developed, not only to aid the exploration we conducted, but also to serve as a multipurpose simulation tool that can be used in similar research areas. The simulation addresses the aspects of performance, power consumption and thermal modelling. To this end, a set of simulation tools has been used, namely, Sniper Simulator [18], McPAT [19] and 3D-ICE [20], [21].

As expected, the framework is able to simulate a multitude of hardware configurations. From a performance perspective, the range of available configurations is that offered by the Sniper Simulator (SniperSim). This range includes but is not limited to, multicore chips that number up to hundreds of cores, with configurable architectural characteristics. It is worth noting that heterogeneous configurations are also supported. In the general case, the other simulation tools used, can easily accommodate such different setups with little or no change, by virtue of the manner in which they are interconnected in the framework. From a thermal point of view, the available options are those supported by 3D-ICE, many package specifications commonly used in current chips. Additionally, a number of configurations were added in the scope of this thesis, by modifying the source code of the tool. However, any further information will not be presented here because the thermal model will be analyzed in detail in section 3.3.

The general flow that is followed during the course of a simulation, is briefly presented in Figure 3.1. It can be readily observed that simulations are performed in intervals, until completion of the requested scenario. This feature, allows us to observe the behavior of the system at fixed time intervals, gather statistics through time, and perform control actions when deemed necessary.

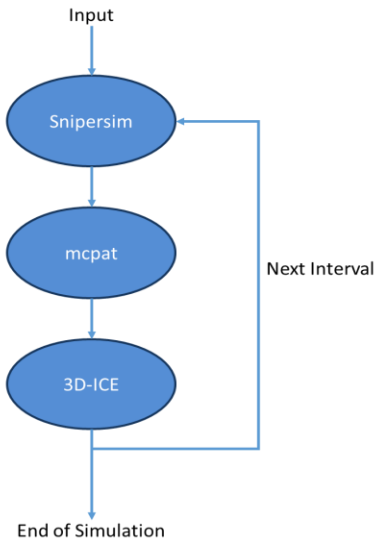


Figure 3.1: Basic framework flow

In the figure presented, the arrows indicating the flow of execution are in fact separate entities, performing a number of operations. The details are not yet outlined for the sake of clarity. All added functionality is implemented in Python [21] as a result of the ability offered by SniperSim to directly control the simulator through the use of Python scripts. This option is enabled by including the *–s* flag in the command line invoking sniper, followed by the script to be used. Any input variables addressed to the Python script mentioned, are listed after the name of the script, preceded by a colon. Scripts that are intended to be invoked in that manner, need to be placed in the *scripts* folder inside the SniperSim installation directory or be specified with a full path.

If we were to include, briefly, some of the intermediate functions implemented in Python scripts, along with their general purpose, the flow presented in Figure 3.2 would ensue. Let it be known that the listing named *Python Interface* essentially denotes the script invoked with the *–s* flag mentioned earlier. Let us name this script *SniperControl.py*. In our case, this script performs a variety of actions, a portion of which, is not directly related with SniperSim. For example, somewhere along the flow of an interval, it is necessary to parse the output values of McPAT and feed them to 3D-ICE. This necessity is only indirectly related with the function of SniperSim. Nevertheless, any action executed outside the normal operation of the simulated tools is invoked and implemented through the use of *SniperControl*.

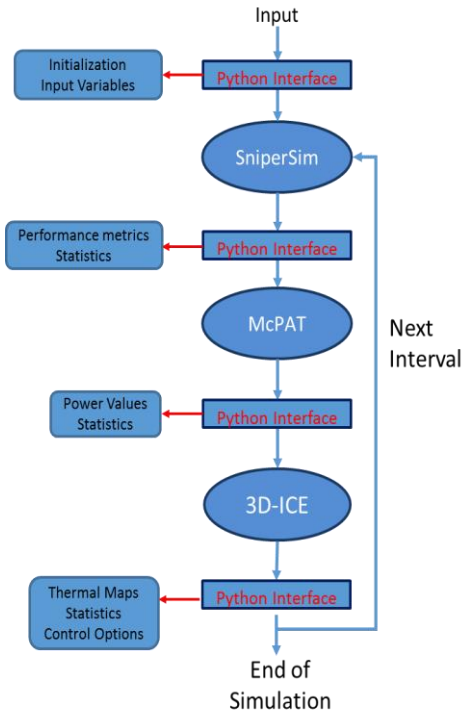


Figure 3.2: More detailed framework flow

To elaborate, a simulation might be summarized as follows:

- SniperSim is called, *SniperControl* is invoked with the *–s* flag
- SniperSim completes the initialization phase for the simulation requested and pauses
- Control passes to *SniperControl* - *SniperControl* performs any initializing action specified and passes control back to SniperSim
- Performance simulation for the specified time window is conducted, SniperSim is paused
- Control passes to *SniperControl* - *SniperControl* calls McPAT for a partial run
- After power consumption simulation for the specified time window is completed, McPAT terminates
- Control passes to *SniperControl* - *SniperControl* calls 3D-ICE client in the appropriate fashion
- After thermal simulation for the specified time window is completed, 3D-ICE client terminates
- Control passes to *SniperControl* - *SniperControl* calls a Python script implementing the policies we would want to enforce in the system. Upon completion, this script terminates
- Control passes to *SniperControl* - *SniperControl* completes any remaining actions, for example, actions regarding logging, and passes control back to SniperSim
- A new time window starts

As we can see from Figure 3.2, there is a small inconsistency regarding the initialization process. In truth, SniperSim is invoked and initialized before it passes control to *SniperControl*. However, for practical purposes, we can safely adopt the simplified model presented in Figure 3.2. In addition, from a stricter point of view, *SniperControl* is not called, as a Python script, in multiple instances. It is in fact called only once, at simulation startup. The control switching that was described earlier, is implemented by registering callback functions. Callback functions, as denoted by their name, are executed by interrupting the simulation each time a certain event occurs (they are called back). Such an event could be the starting of a thread, or the passage of a certain amount of simulated time. While various callback functions are used in our Python Interface, we perform the majority of our control actions every time a fixed value of simulated time has elapsed. The flow that was presented is addressed to this particular kind of control switching and generally abstracts away other callback functions, since they are not directly relevant to the overall operation of the system and are mainly used for exporting statistics and logging. More information regarding callback functions and how they are used in SniperSim, can be found in the documentation of the Sniper Simulator [17].

Returning to the analysis of the more detailed model, we can see that a call to the Python implemented interface, precedes the operation of each simulation tool. In a general sense, the main use of each such call, is to export statistics regarding the simulation output of the previous level,

gather the necessary input variables for the next tool to be used, format them in a suitable form, and pass them along with the call to the appropriate program. A special case in this pattern, is the first call to the Python Interface where no statistics are exported. In this case, only a set of initialization actions is performed. Furthermore, the input variables gathered in this instance are only those addressed to *SniperControl*. The input variables that are meant for the Sniper Simulator are fed to it directly, through another entity.

When a single and isolated instance of SniperSim is considered, the tool can be called along with its configuration variables through the terminal, using a command line. However, when multiple simulations, with different configurations, are intended to be explored, passing input variables in this manner can be quite tedious. Moreover, it necessitates waiting for each simulation to end before being able to request another. These problems arise when SniperSim is used exclusively. For our framework, which also includes other simulations tools, this approach presents yet other problems.

In order to use 3D-ICE for interval simulation, we implemented a Client –Server model, as described in the documentation of the thermal simulator. All configuration parameters and structures pertaining to the thermal model, along with the thermal status of the system, are maintained in the server side. All computation relevant to the thermal simulation is also performed by the server. In truth, the client only serves as an intermediate communication interface between the Python Interface and the 3D-ICE Server. Communications exchanged, mainly involve input values for each interval and thermal maps describing the resulting thermal status of the chip stack.

While the client is called whenever power inputs for the server are available, in an on demand basis, and then terminates, it is quite obvious that the server must remain online for the entire duration of the simulation. Furthermore, between simulations, the server must exit and restart with a new configuration set. Essentially, a server complete run corresponds to a full simulation while a client run corresponds to a time interval. This model we adopted, requires the manual initiation of the server prior to the beginning of each simulation. Moreover, the server initialization is associated with a number of files that describe the configuration of the chip.

It is essential for the function of the thermal simulator that a stack description and a floorplan file are created and placed properly. These files also contain all configuration parameters of the thermal model. Manually creating and modifying these input files, based on each scenario, is not an effective solution. An obvious choice for accomplishing such a task, and the activation of the server, would be *SniperControl*. However, this choice was avoided.

Instead, a higher level entity was created. The factors that paved the way for this approach will slowly unfold as we describe the function of this higher level module. This structure, implements a Client-Server model, developed in Python. It serves as an intermediate between the simulation framework and the user. The input to this module, is a series of simulations to be performed. The server side basically handles starting and exiting the 3D-ICE server prior and after each simulation, respectively. The client side handles all pre-simulation necessary actions including the creation and placement of stack description and floorplan files according to the requested specifications.

In this way, the Python server needs to be started only once, and then serves simulation requests indefinitely. The Python client enables the user to specify a queue of simulations to be run and facilitates the input of configuration variables for all simulation tools in one unified interface. A graphical representation of this model can been seen in Figure 3.3. This is in fact a macroscopic view of the simulation framework, meaning, that the parts analyzed previously, and graphically illustrated in Figures 3.1,3.2, are integral, lower level parts of the Python client side.

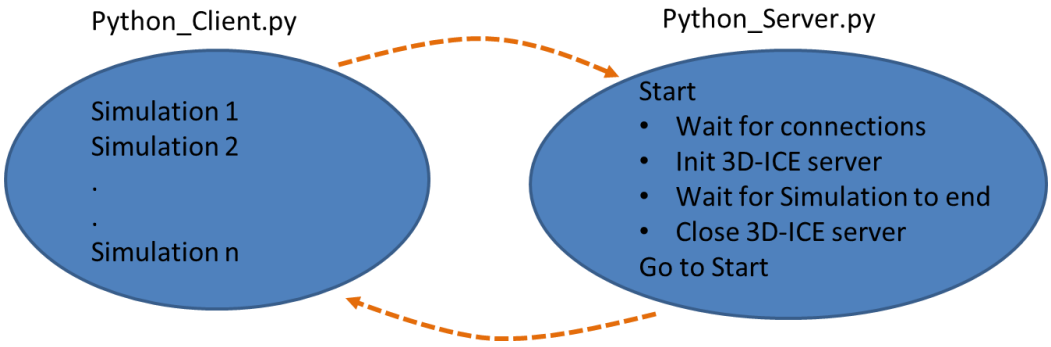


Figure 3.3: Python client-server model

We intentionally built our way towards this less detailed representation, in order to provide a better understanding of the process that took place in combining the simulation tools into a unified whole, and the benefits of our approach. Figure 3.4 is a graphical summary of what was described so far. After inputting a set of simulation requests and running the appropriate script, the flow in the mentioned figure is followed. *Simulation_Queue.py* will initialize all configuration files as per ordered. When initialization is complete, it will communicate to the server to start a 3D-ICE server session. The server will report back after the 3D-ICE server is up and running, granted that no error occurs, and *Simulation_Queue* will continue by calling SniperSim with the appropriate variables. Afterwards, until completion of the simulation, the interval flow that was described earlier follows. This flow is now represented as an implied, separate module, framed by a yellow dashed box. Regarding the function of the thermal model, the server – client function that was analyzed, is now schematically outlined in "magnification" and placed aside in order to maintain the conceptual integrity presented previously. When power values are available, the 3D-ICE client is started, communicates them to the 3D-ICE server, receives the results and terminates. At the end of the simulation, ownership is transferred again to *Simulation_Queue*, which in turn communicates to the Python server to end the current 3D-ICE server session. The next simulation characteristics are picked from the list and a new identical loop begins.

From a user level-perspective, only the uppermost input interface is visible and needs to be edited in order to issue a list of simulations, diverse in performance and/or thermal characteristics. In order to facilitate our work, only those variables that we most commonly altered were propagated to the highest level allowing to be readily edited. Nevertheless, the bulk of the configuration variables of any simulation tool used, can be easily exported to this first level

interface. Additionally, before any simulation is requested, the Python server must be started. In our case, this server was launched manually and was tied to a terminal window in order to assist the development of the framework. For a system intending to be used for simulations, the server can be easily started as a background process, at system startup or on demand, and probed only when deemed necessary.

As far as the export of the client side is concerned, each running simulation keeps all files in a temporary folder, named *Current_Simulation*. Files included in this folder are outputs of the simulation tools along with the standard output and standard error for each, utility files used for various purposes and will be analyzed is subsequent sections, configuration files used, and other logging material. This temporary file is wiped at the end of each simulation, after all the files are properly archived, in order to be used by the next request in queue. If any kind of error arises during the archiving process, the simulation framework is halted, so that the user can manually extract the simulation files and prevent data loss. A number of options on how to archive each simulation is available. Moreover, adding new ways to store requested simulations is feasible by adding a function to the corresponding Python script.

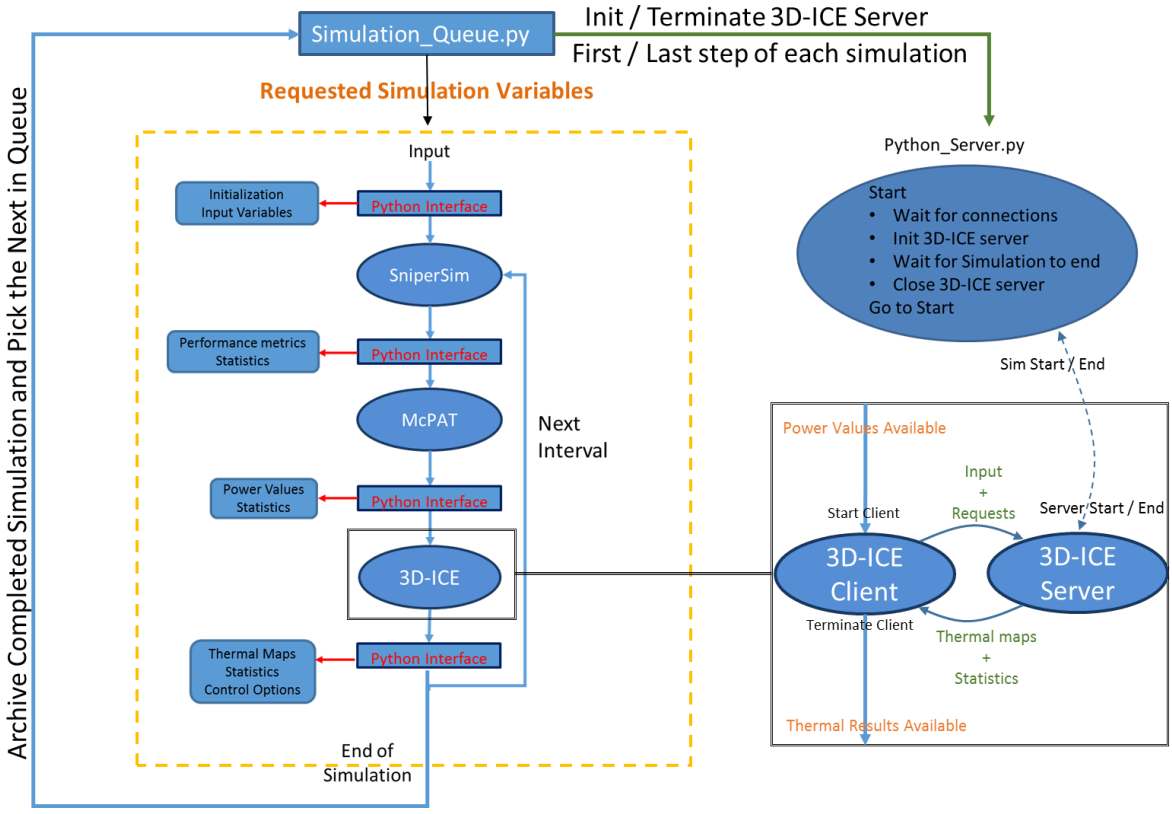


Figure 3.4: Framework Summary

Regarding output files and statistics, a number of metrics are by default exported during a simulation. Even though not all of them were ultimately used in our work, we made a selection based on the metrics most commonly used. Even so, we refrained from outputting a very big

number of variables in order to keep simulation output files relatively small in memory size. Statistic variables regarding performance and thermal values are a special case in our framework. In contrast with our general approach, exporting a new statistic is probably not a straightforward, nor simple task. It would require a direct modification to the corresponding interface of each tool. In the case of SniperSim a number of variables might be rather easily accessible by adding their entry to *SniperControl*. However, in some cases, it might be necessary to modify the source files of the simulator. For 3D-ICE, adding new outputs of any form, would require adding requests and/or other complementary code in the 3D-ICE client and/or 3D-ICE server.

As is usually the case, the development of this framework was not smooth and error-free. Before this presented form, the module has underwent a deal of restructuring and a number of revisions. Even in this fully functional version, a number of improvements and additions that can be implemented are specified and will be presented in the proposed future work. Note also, that the approach we followed in designing this framework is not exclusive. In various places, many alternative ways to achieve the same result were available. Arguably, some of them might have been more effective in terms of performance and/or other characteristics. For example, instead of developing the Python server to handle the 3D-ICE server, we could have modified the source code of the latter to accommodate multiple simulations, that is, to dispose all structures at the end of a simulation and parse new floorplan and stack description files. However, a performance and optimization exploration, is outside the scope of this thesis. While in future versions, or if, hypothetically, we developed the framework anew, we would certainly redesign some modules, the approach that was chosen in each case, reflects an effort to balance efficiency and development time, with respect to the fields that we were more accustomed to.

In summary, the simple use of the framework involves only starting the Python server once, filling out characteristics for the requested simulations to be performed in the appropriate script, and running it. Even though the framework in not ideal, in order for it to be adaptable to the custom needs of each research approach, and to any kind of modification of addition, after this rather abstract overview, comes a detailed description of most of the interconnecting parts. Obviously, the preceding macroscopic presentation is essential to the understanding of the subsequent section. Rather than keeping the big picture into perspective, section 3.2 will analyze each tool and relevant scripts in isolation and then conclude into a unified, fully-detailed figure. Lastly, a number of small inconsistencies might be observed regarding functions that are assigned ultimately to lower-level modules. This is a direct result of the effort to present a more compact figure in the overview section, easier to understand, and not an oversight.

3.2 Python Interface Detailed Analysis – Basic Scripts

3.2.1 Simulation_Queue.py

The top-most script of the simulation framework. A number of lists are declared in the first lines of this file, where the user can fill out the characteristics of the simulations that are meant to be run. The values specified are unpacked in the order they are written in the script. That is, the first simulation will be performed with a set containing the first value from each list, the second with the second value from each list and so on. *Simulation_Queue*, when run, immediately groups the corresponding values and calls *Sniper_Simulation_Init.py* with those values as input. The input values mentioned, are not necessarily a single item. Some per-simulation variables are defined by a list (like the elements in the chip stack) so the resulting module in the script is a list of lists. Each top-most list completed must have the same length, one value per simulation to be run. For variables that do not change from simulation to simulation, it is possible to just multiply them with the number of simulations to be run, rather than having to type them one by one. All values defined in any list, have to be represented as a string. A more detailed analysis of this script and the variables included will be demonstrated in section 3.5.

3.2.2 Sniper_Simulation_Init.py

This script performs all necessary actions, at the beginning and end of each simulation execution. It does so, utilizing a script named *Simulation_Utilities.py*. In general, the Python Interface is arranged in a high to low level approach, distinguishing basic and utilities scripts, in an effort to make the layout more structured. The functions included in *Simulation_Utilities* are:

- **Configure_Variables:** Defines a number of variables that will be used throughout the execution of the script. The variables are added into a dictionary that is returned to the basic script.
- **Initialize_Sim_Folder:** Completely wipes the *Current_Simulation* directory from all files and folders to prevent a misplaced file or a failed execution to create unforeseen results. Subsequently, it creates the necessary sub-directories for the simulation.
- **Write_Config_File:** Creates a configuration text file describing all the parameters of the requested simulation.
- **Initialize_Simulation_Files:** From a directory of already created floorplans, locates and fetches the appropriate floorplan file for the requested configuration. In addition, a corresponding visual representation, meaning an image showing how the components are arranged is also fetched. This image will be used by another, optional, post-simulation script that is able to visualize thermal maps. Implementing another script named *STK_Utils.py*, it creates the stack description file with respect to the corresponding parameters requested. If a phase change material is requested, by use of *PCM_Classes.py* it also adds a corresponding entry to the stk file, and creates a PCM description file with

the appropriate variables. The PCM description files are parsed from the 3D-ICE server as a result of the added capabilities we have implemented to the thermal simulator. All files are placed in the *Simulation_Files* folder.

- **Initialize_Server:** Communicates with the Python server, requests the initialization of the thermal simulation server, and waits for reply. When the latter is started without error, the Python server communicates the fact, along with the socket it listens to.
- **Start_SniperSim:** Runs the sniper simulator along with all the corresponding command line arguments.
- **Post_Simulation_Actions:** It is executed after the simulation is completed and signals to the thermal simulation server to terminate. Next, it archives the simulation that was performed using a function included in a script named *Archive_Utils.py*. This script contains a number of functions defining ways to create the directory where the simulation files will be stored to provide greater flexibility (e.g. by date or by architecture or any other combination of inputted variables and constants). If the function is successful, *Current_Simulation* is wiped.
- **Reset_Current_Sim_Folder:** When called, completely erases *Current_Simulation* folder and all its subdirectories. Before exiting, recreates a new empty folder with the same name.

```
import sys

sys.path.insert(0,'/home/Simulation/'
                  'Python Scripts/Utils/'
                )

from Simulation_Utilities import *

args=dict(enumerate(sys.argv))

Variables=Configure_Variables(args)

Initialize_Sim_Folder()

Variables=Write_Config_File(Variables)

Initialize_Simulation_Files(Variables)

Variables=Initialize_Server(Variables)

Start_Sniper(Variables)

Post_Sim_Actions(Variables)
```

(a)

```
import os,shutil,fileinput,zmq,subprocess,glob,time,math
from Archive_Utils import *
from Stk_Utils import *
from Pcm_Classes import *

def Configure_Variables(args):


def Initialize_Sim_Folder():
    simulation_folder='/home/Simulation/Current_Simulation/'
    directories=['snipersim','visual_output','thermal_maps','power_map',
                 'simulation_files','misc','simulation_stats'
                ]

    Reset_Current_Sim_Folder()

    directory_file_names=os.listdir(simulation_folder)
    for file_name in directories:
        if file_name not in directory_file_names:
            os.mkdir(simulation_folder+file_name)

def Write_Config_File(A):

def Initialize_Simulation_Files(A):

def Initialize_Server(A):

def Start_SniperSim(A):
    os.system('/home/sniper-6.1/benchmarks/run-sniper '
              ' -n %s -s %s -d %s -c %s --benchmarks=%s --sim-end=last 2>&1'
              '| tee /home/Simulation/Current_Simulation/misc/screen_log.txt'
              %(A['core_number'],A['script'],A['sniper_dir'],A['architecture'],
                A['benchmark'])
             )

def Post_Simulation_Actions(A):

def Reset_Current_Sim_Folder():
```

(b)

Figure 3.5: Example of a basic and utility script:
(a) Sniper_Simulation_Init (b) Sniper_Simulation_Utilities.
In (b) an example of functions is shown and the others are folded.

3.2.3 Sniper_Simulation_Control.py

This script is the cornerstone of the Python Interface. It is the same script we temporarily named *SniperControl* in the previous overview. In contrast with almost all the other scripts, *Sniper_Simulation_Control* employs mainly the use of classes to register objects to the simulator. Within these classes callback functions are defined. These functions are executed by interrupting the Sniper Simulator whenever a certain event occurs. We will focus our attention to the periodic function included in the class *Simulation_Flow*. This function is called every time the user-specified interval has elapsed. The first step of this function is to call McPAT with the current time interval as input, for a partial run. To implement this call, another function named *McPAT_Partial* is employed. In order to accommodate DVFS controls during simulation runtime, at each interval a configuration file containing the frequency and voltage level of each core is produced, and passed with the *–c* flag to McPAT. More about the input variables that can be defined when calling McPAT, and other options, can be found in the tool documentation.

After this call returns, the script employs the use of the function *Thermal_Simulation_Client.main* that is included in *Thermal_Simulation_Client.py*. The variables passed to this function are the two time values designating the beginning and end of the current time interval. Although it is employed here as a function, *Thermal_Simulation_Client* is a basic script and will be further analyzed in the following section.

In brief, in accordance to our earlier analysis, *Thermal_Simulation_Client* parses the output of McPAT and calls the thermal simulation client with the power values obtained. Once the simulation is complete, the client, as well as the script, terminate, and control is again returned to *Sniper_Simulation_Control*.

Afterwards, another basic script is called in the form of a function, namely, *Resource_Control.py* as *Resource_Control.main*. This function-script, takes as input the time interval, the number of cores in the sim, and the frequency levels of each core in the form of a list. In our work, *Resource_Control* does not have a stable role. It has access to all metrics of the simulation and can accordingly make resource allocation decisions. For example, change the frequency-voltage levels of a core. In general, any allocation scheme can be employed here. This script will be detailed more thoroughly afterwards.

Lastly, following the decisions described by *Resource_Control*, a function named *Resource_Control _Sim* is called in order to implement everything specified by communicating with the simulator. In our case, a simple voltage-frequency level allocation is enforced, accompanied by the production of a configuration file that will be used as input to McPAT in the next interval.

Note, that the *Simulation_Flow* class we just described is the only one defined in *Sniper_Simulation_Control* due to its pivotal role. Other classes that are registered through this script to the simulator, are included in a file named *Sniper_Simulation_Classes.py*. The classes defined there are the following:

- **Performance_Statistics:** Logs the value of a number of metrics through time. In our work the metrics logged were: ipc, cycles, instructions, L1-D misses, LI-I misses, L2 misses and L3 misses.
- **Thread**: A wrapper class for manipulation of threads. Only sets a number of thread-specific variables.
- **Thread_Events:** Logs all thread events using the *Thread* class. The events logged are: create, start, stall, migrate, resume and exit, along with the timestamp of the event.

Furthermore, classes defined in both scripts, utilize functions included in *Sniper_Simulation_Utils.py*. The functions listed there are:

- **Resource_Control_Sim:** Takes a frequency table as input and communicates only the values that changed from the previous interval to the simulator. Obviously, the simulator sets the requested levels at the beginning of the next interval. This function also produces corresponding voltage levels using *Build_DVFS_Table*, and logs them with the frequencies in a config file that will be used by McPAT in the next iteration. The configuration file is produced by the function *Generate_Config*.
- **Build_DVFS_Table:** Builds a voltage-frequency table based on pre-defined voltage-frequency pairs.
- **Generate_Configuration:** Produces a configuration file with the appropriate name (the time interval it addresses), listing in the correct format for McPAT the voltage-frequency pairs of each core.

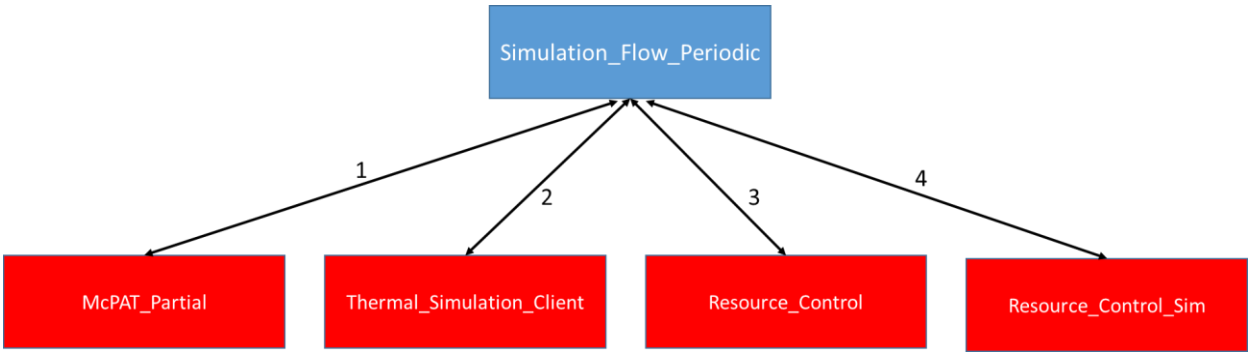


Figure 3.6: Simulation_Flow Summary

3.2.4 Thermal_Simulation_Client.py

This script handles all necessary actions related to the communication of the Python Interface with the thermal simulator server through the use of the corresponding client. *Thermal_Simulation_Client* uses a set of functions included in the script *Thermal_Simulation_Utils.py*. In detail:

- **Configure_Variables:** Gathers a set of necessary variables for the function of the basic and utility script and structures them in a dictionary that is returned back.
- **Pre_Computation_Actions:** Initializes a set of text files that will be used to record statistics.
- **Power_Values_Computation:** Parses the McPAT output text file for the current time interval and computes the power value of each component. The computation involves locating in the text file, each element used in the floorplan, adding the separate power values listed, recording the sum along with other metrics to the appropriate text files, and storing it to a list. The power values that were added are: Subthreshold Leakage, Gate Leakage and Runtime Dynamic. To handle these steps that result in a list containing the power values of each component, another script named *McPAT_Parse.py* was used.
- **Post_Computation_Actions:** Records the list of power consumptions that were computed along with metrics that utilize aggregate values, e.g. total power consumption or max power density, and records them.
- **Thermal_Simulation_Client_Call:** Calls the 3D-ICE client with the appropriate command line, passing along the computed power values and the thermal simulator server socket. The standard output and standard error of the program are captured in likely named text files.

3.2.5 Resource_Control.py

This script is the proper place in the interface to accommodate any resource management scheme. A simple sprint until exhaustion policy would just have to produce a frequency table with the values for each core and return it upon completion. *Sniper_Simulation_Control* will enforce the values requested to the simulation. In order to implement reactive control when temperature exceeds a certain limit, a function monitoring the max chip temperature is sufficient. It stands to reason that when a violation occurs, the function would override the sprinting policy and issue a frequency table with sustained execution values. This table is the one that should be now returned to *Sniper_Simulation_Control*. In the case that a completely different policy in intended to be used, another high-level script could be produced. Let us name this script *Resource_Control2.py*. The only modification needed, would be to invoke *Resource_Control2.main* instead of *Resource_Control_main*, in *Sniper_Simulation_Control*. Every script that might be produced in this fashion, can opt to use functions included in *Resource_Control_Utils.py*. In our work, this script includes the following:

- **Configure Variables:** As always, gathers a number of necessary variables and structures them in a dictionary returned to the basic script.
- **Parse_Core_Temps:** Parses the corresponding file outputted from 3D-ICE which contains the max temperature values for each core. These values are available through the extra capabilities we integrated in the thermal simulator in the form of small additional code segments. No special examination of these parts is deemed necessary. Obviously, the values that are parsed are added to the variable dictionary.

- **Form_Violation_Matrix:** Creates a matrix listing whether a core has reported a temperature violation. This matrix was used in order to test sprinting policies that use per-core activation and deactivation.
- **Frequency_Allocation:** Decides on the target frequency allocated to each core based on whatever criteria we impose. In our work we mainly steered this function into choosing a single, stable, above sustained levels frequency.
- **Violation_Policy:** Implements a reactive control policy when a violation is observed. In our work this policy would either revert all cores to sustained execution or revert each core reporting a violation.
- **Decision_Log:** Writes to a suitable text file all decisions taken at each interval.

In addition to the functions described, *Resource_Control_Utils* contains others, not used in the presented example. The idea was to have a single utility file facilitating the creation of a number of basic scripts implementing different allocation policies.

3.2.6 Python_Server.py

The Python Server is implemented by using ZeroMQ [22] in addition to the rest of the code. In short, ZeroMQ is a high-performance asynchronous messaging library supporting many programming languages, including Python. The basic script *Python_Server.py* includes functions from *Python_Server_Utils.py*. Listing them along with a small description will also explain the use of the server script:

- **Configure Variables:** A function we have seen in almost all basic scripts with basically the same role.
- **Initialize_Python_Server:** As denoted by the name, initiates the ZeroMQ module and binds a socket whose value is stored in the variable dictionary.
- **Wait_For_Simulation:** The server listens to the previously bind socket for messages. If a proper header is detected, the rest of the message is parsed as the requested configuration and execution resumes.
- **Initialize_3DICE_Server:** Follows the detection of a proper simulation request. Starts the 3D-ICE server with the proper stack description, floorplan and other files, and sends an appropriate message to the Python client when the 3D-ICE server finishes the initialization process (barring any error).
- **Accept_Connections_and_Close:** Logs all 3D-ICE server output for the duration of the simulation. When the simulation is completed, this function terminates and the basic script goes back to Wait_for_Simulation.

3.2.7 Low Level Scripts

This section contains scripts that are part of the Python interface but are not readily observed. They implement specific functions for utility files included in basic scripts. In essence,

they could be named sub-utility files or even-lower-level scripts. A brief presentation of the operation of each of these scripts will follow:

- **Archive_Utils.py:** This script is imported in *Simulation_Utilities*. It contains functions, each of which, defines a way to archive completed simulations. An example would be storing them based on benchmark, worksize, and number of threads, that is *outputs/benchmark_name/worksize/thread_number/files*. The idea is to specify in *Simulation_Utilities*, the name of the function that we intend to use to store the files, depending on the series of simulations we want to run. In our work, this file included basically a function to store based on benchmark, as above, a function to store based on date, and a function to store based on thermal characteristics.
- **McPAT_Parse.py:** Facilitates the parsing of the McPAT output text files. It is used by *Thermal_Simulation_Utils*. It contains a function addressing the differences in component naming in the floorplan file, from where each component is selected, in the order they are written, and a function that implements the parsing per se, also logging the values along with some derivatives.
- **PCM_Class.py:** A small file included in *Simulation_Utilities*, containing a module for saving the characteristics of a phase change material. The PCM class is used to create a file that will be parsed by the thermal simulation server in order to obtain the extra properties of the phase change material.
- **STK_Classes.py:** Contains a number of classes defining objects used in the creation of a stack description file. The classes defined are: Material, Materials_Used, Layer, Layers_Used, Die, HeatSink, Dimensions, Stack, Solver and Output. All these objects are used in *STK_Utils.py*.
- **STK_Utils.py:** Imported by *Simulation Utilities* in order to create the stack description file that will be used by the thermal simulation server. With the use of classes included in *STK_Classes*, it registers materials, layers and all other necessary entities involved in thermal simulation. Of course, the stack description file is created and properly placed before the script terminates.

All of the scripts mentioned in section 3.2 are summarized in Figure 3.7. The flow is presented in full detail. In addition, any control switchback involved is represented and any multiple instance of a script designates that the repeated script has regained control of the flow. Python scripts names are colored in shades of green while the simulation tools use black. Scripts listed at the right side are lower levels of utility files arranged in descending order. From the structure of the figure, it is visible which script uses which utility script.

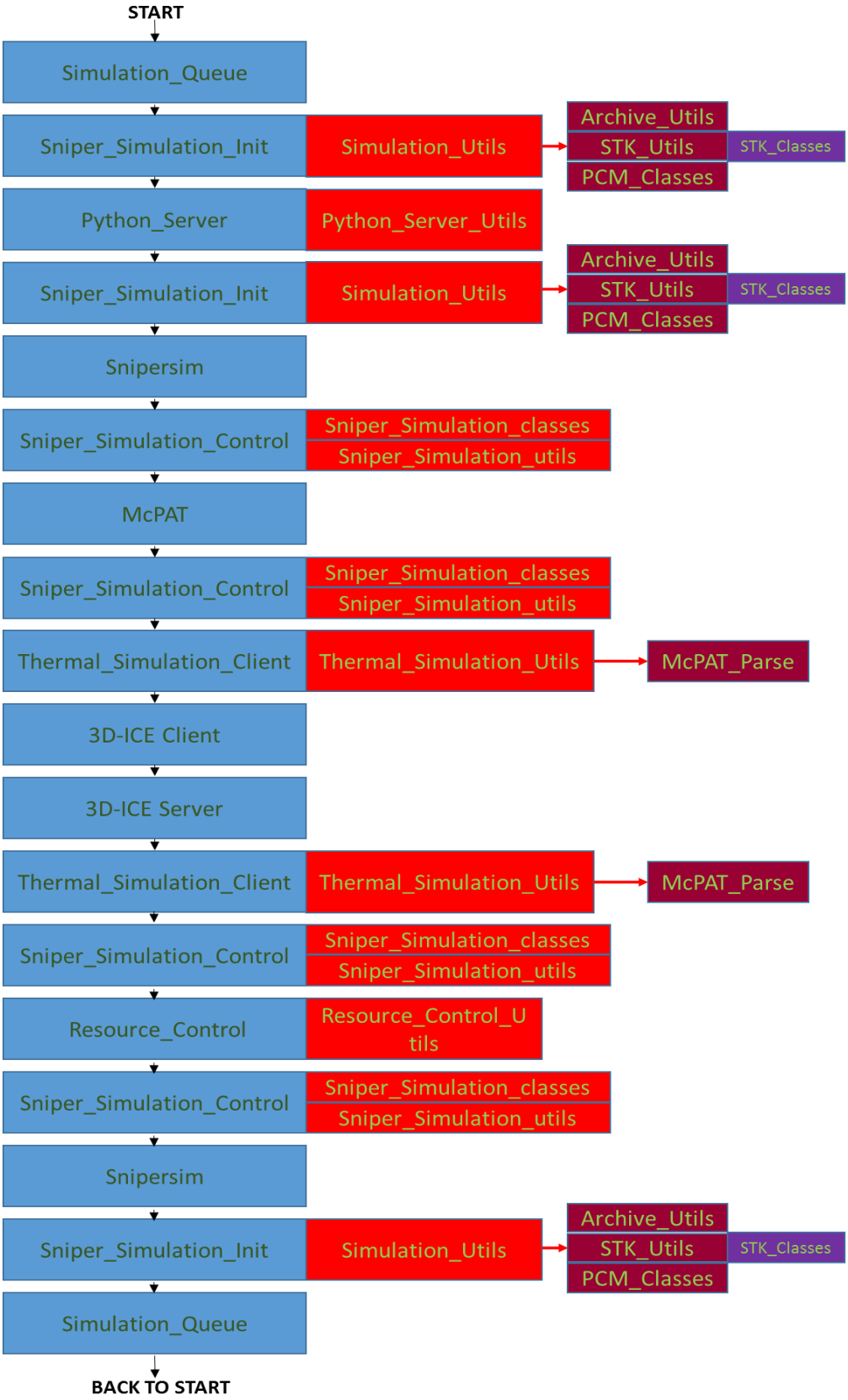


Figure 3.7: Python Interface summary

3.3 Thermal Model

3.3.1 3D-ICE Heat Conduction Modelling

The modelling of heat conduction in solids in 3D-ICE is done by applying finite-difference approximation to the governing equations of heat transfer. The exact process is described in [19]. In the last step of this process, the well-known analogy between heat and electrical conduction is invoked. That is, temperature is represented as voltage, heat flow is represented as electric current. Thermal conductance and resistance are replaced by electrical conductance and resistance respectively. Lastly, heat capacity is substituted by electrical capacitance.

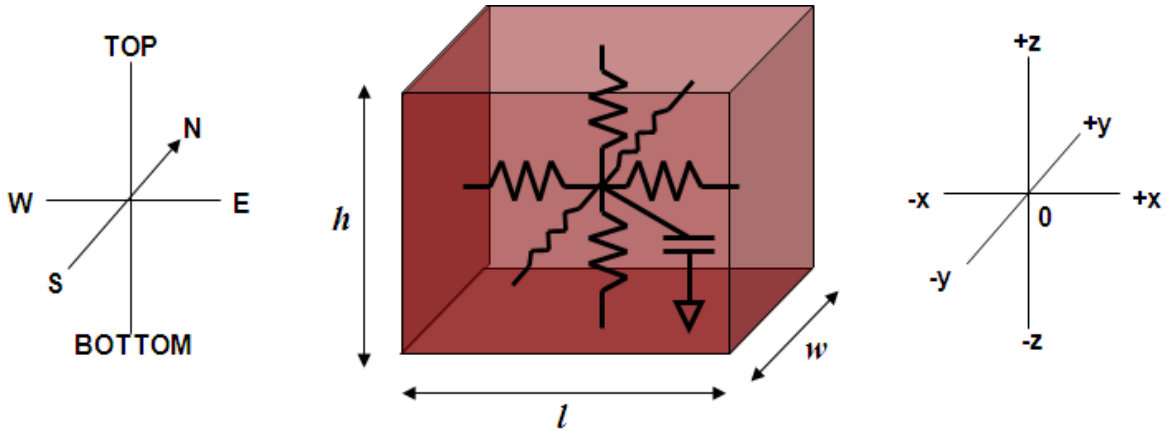


Figure 3.8: A typical solid thermal cell

Afterwards, the thermal model is generated considering each layer to be discretized into cuboid thermal cells based on user-defined discretization parameters. Each thermal cell has a length *l*, width *w*, and height *h*, as shown in Figure 3.8, modeled as a node containing six resistances representing the conduction of heat in all the six directions (top, bottom, north, south, east and west), and a capacitance representing the heat storage inside the cell. The conductance of each resistor and the capacitance of the thermal cell are calculated as follows:

$$g_{top/bottom} = k_{S_i} \cdot \frac{l \cdot w}{(h/2)} \quad , \quad g_{north/south} = k_{S_i} \cdot \frac{l \cdot h}{(w/2)} \, ,$$
$$g_{east/west} = k_{S_i} \cdot \frac{w \cdot h}{(l/2)} \quad , \quad c_{cell} = C_{vS_i} \cdot (l \cdot w \cdot h) \, . \tag{1}$$

The subscripts *top*, *east*, *south* etc. indicate the direction of conduction (i.e. north represents conduction in the +y direction, west represents conduction in the –x direction and so on). Current sources representing sources of heat are connected to the cells wherever there is heat dissipation. Next, the nodes of these thermal cells are connected to the nodes of their neighboring cells through

the interfaces by computing the equivalent conductances between them. Hence, the following system of ordinary differential equations is created:

$$\boldsymbol{GT}(t) + \boldsymbol{C\dot{T}}(t) = \boldsymbol{U}\ (t), \tag{2}$$

Where **T**(t) is the vector of all node temperatures (as a function of time), **C** is a diagonal matrix of all cell capacitances calculated using Equation (1), **U**(t) is a vector of inputs (heat sources as a function of time) wherever they exist. **G** is a symmetric block tri-diagonal conductance matrix where non-zero, non-diagonal elements represent the connections between neighboring nodes and the diagonal term corresponding to a given node is equal to the sum of all conductances between that node and its neighbors.

The formulation of heat flow equations, as described above, can be extended to structures containing multiple layers of thermal cells. This method can be used to generate a compact thermal model for any general heterogeneous structure like an IC die, and the three-dimensional temporal evolution of heat inside the 3D-IC, can be accurately modeled. In order to formulate the equations for the simulation of the thermal grid, Equation (2) is integrated numerically using the backward Euler method as follows:

$$\begin{gathered}\left(\mathbf{G} + \frac{1}{h}\mathbf{C}\right)\mathbf{X}(t_{n+1}) = \mathbf{U}(t_{n+1}) + \frac{1}{h}\mathbf{C}\mathbf{X}(t_n) \\ \Rightarrow\ \mathbf{A}\mathbf{X}(t_{n+1}) = \mathbf{B}_{n+1}\end{gathered} \tag{3}$$

Where h is the time-step of the numerical integration, $\mathbf{A} = \mathbf{G} + \frac{1}{h}\mathbf{C}$, and $\mathbf{B}_{n+1} = \mathbf{U}(t_{n+1}) + \frac{1}{h}\mathbf{C}\mathbf{X}(t_n)$. Here, t_n denotes the n^{th} time point during the transient simulation. In order to acquire the necessary variables to formulate the thermal model, 3D-ICE needs the following inputs:

a) The physical description of the IC layers comprising the chip stack and their material properties.
b) The discretization parameters (thermal cell size, time-step etc.) along with the chip size and initial temperature of thermal cells.
c) Floorplan information of each individual die, reflecting location and area of various circuit blocks. These blocks will later have power dissipation values assigned to them.

3D-ICE is a software thermal library built in C and based on the thermal modelling we just analyzed. All of the values listed previously are given to the simulator via netlist files. The netlists are parsed and the matrices **A** and **B** are generated. As we can easily conclude according to the preceding analysis, matrix **A** is constant during a simulation and therefore is calculated only once. On the contrary, matrix **B** is dependent both on the previous temperature values and the current heat sources. As a consequence, it is recalculated at every time step before solving the sparse linear system.

In addition, as mentioned in [23], the governing equations presented in the analysis in [19] can not only be used for solids, but also for liquids and gases that are considered to be stationary. In our work this is of particular interest because we intend to model phase change materials. Specifically, this fact allows us to retain the same model even after the PCM thermal cells are melted, meaning, they are in the liquid phase, because we consider them to be unassociated with any type of movement.

3.3.2 PCM Modelling in 3D-ICE

In order to model phase change materials in 3D-ICE, we used the *apparent heat capacity* method from [9]. In this method, a nonlinear temperature dependent specific heat capacity is assigned to the PCM layer as shown in Figure 3.9. The transition of the phase change material from solid to liquid occurs over a temperature interval, where the specific heat capacity is very high compared to the material's heat capacity in the solid and liquid phases. However, as can be seen in the function described in the figure, the transition of the heat capacity is not instantaneous. It rises from a value characterizing the solid phase at a steady rate (linear region), then assumes a maximum value for a certain duration and lastly decreases at a steady (equal to the previous) rate up to the value characterizing the liquid phase. Due to the increase in specific heat capacity, the rate of change of temperature decreases during phase transition.

In real situations, during change of phase, a material absorbs large amounts of energy at approximately stable temperature. The behavior that results from altering the specific heat capacity dynamically during runtime, in the way specified, simulates real phase transitions very accurately. The integral of the heat capacity over the transition temperature range equals the latent heat of fusion for the PCM. This fact leads to Equation (4) used in calculating the max heat capacity for the method relative to the latent heat of fusion which is an intrinsic material property. Recall that this property was specified in chapter 2.

$$\int_{T_1}^{T_2} f(T, C_{max}) dT = Latent\ Heat\ of\ Fusion \tag{4}$$

In order to explain how we implemented the apparent heat capacity method in 3D-ICE, we need to first go through a quick overview of how the simulator normally operates. Disregarding all other features except those important to our purpose, we can summarize the execution flow of the simulator as follows:

1. Parsing of all input files, storing of variables in appropriate structures
2. Initialization of the structures to be used next (memory allocation etc.)
3. Formulation of matrix **A** , with respect to the input values (thermal configuration)
4. Wait for connections
5. Connection made, power values for the next interval are obtained
6. Computation of matrix **B**
7. Solve sparse linear system

8. Repeat numbers 6-7 for the number of specified time steps (iterations, each with the temperature values resulting from the previous step)
9. Write results to thermal maps
10. Go back to 4

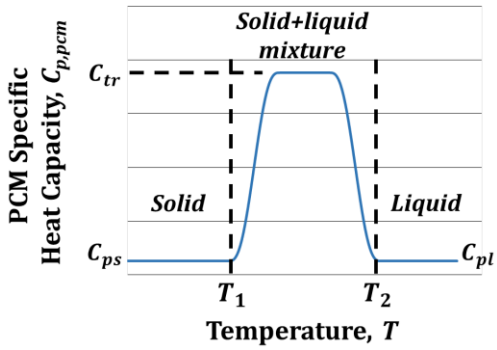


Figure 3.9: Piecewise linear function for PCM specific heat capacity.
Setting $c_{tr} \gg c_{ps}$ for the (T1, T2) interval models the phase change

As we can see, the computation of matrix **A,** happens only once for the entire duration of a simulation. This happens because, as we explained earlier, the values relative to its computation are stable, defined by the thermal configuration and known after parsing the netlist files. Matrix **B** by comparison, is seen in Equation (3) to be dependent on the temperature and power values of each interval and is therefore recomputed. Needless to say, since the method used to derive Equation (3) is a numerical one, a number of steps is required in order for the computed values to converge to the "exact" solution. From the documentation of 3D-ICE, a number of 5 steps is reported to be sufficient. If we were to integrate the apparent heat capacity method to the flow described, the conditional in listing 3.1 would ensue.

Listing 3.1: Conditional describing the apparent heat capacity method in pseudo-code

```
IF ( CURRENT_CELL → PCM_LAYER)  AND
   (CURRENT_CELL_TEMP > T1)       AND
   (CURRENT_CELL_TEMP < T2)          :
         ASSIGN_SPECIAL_C_VALUE
         CONTINUE_NORMAL_EXECUTION
ELSE:
         NORMAL_EXECUTION
```

Obviously, this conditional has only conceptual value. It illustrates the idea behind the specific implementation. In simple terms, the notion is that every time the simulator is looking to fill matrixes with capacitances, cells that belong to the PCM layer and have temperatures within the region of phase change, will be assigned a capacity value simulating the behavior we see in Figure 3.9. In any other case, cells will be assigned values with respect to the layer they belong

and the material the layer is made of. To be more thorough, let us examine an excerpt of code, describing the computation of matrix **B**.

In listing 3.2, we can see that the computation performed is the same described in Equation (3) for matrix **B**. The matrix *sources* contains the power value allocation at the cell grid (matrix **U**), and the matrix *temperatures* contains the temperature values corresponding to the previous interval (matrix $\mathbf{X}(t_n)$). The function *get_capacity* calculates the capacitance of the current cell according to Equation (1) using the dimensions of the cell and the volumetric heat capacity of the material. The role of the matrix *vector* is obvious. In order to implement the method we described earlier, we replaced the code inside the for-loops shown in 3.2 with the code in listing 3.3.

Listing 3.2: Part of the code computing matrix **B**

```
FOR_EVERY_LAYER (layer, dimensions)
{
    FOR_EVERY_ROW (row, dimensions)
    {
        FOR_EVERY_COLUMN (column, dimensions)
        {
            *vector++ = *sources++
                      +
                      (
                      get_capacity (thermal_grid, dimensions, layer, row, column)
                      /
                      step_time
                      )
                      *
                      *temperatures++ ;
        } // FOR_EVERY_COLUMN
    } // FOR_EVERY_ROW
} // FOR_EVERY_LAYER
```

Listing 3.3: Modified Code to compute matrix **B**

```
FOR_EVERY_LAYER (layer, dimensions)
{
    FOR_EVERY_ROW (row, dimensions)
    {
        FOR_EVERY_COLUMN (column, dimensions)
        {
            if (
                (strcmp(thermal_grid->material_id[layer],"PCM")==0)
             && (*temperatures>=(thermal_grid->PCM_Melting_Point-(thermal_grid->PCM_Melting_Duration/2)))
             && (*temperatures<=(thermal_grid->PCM_Melting_Point+(thermal_grid->PCM_Melting_Duration/2)))
               )
            {
              PCM_Region=1;

              *vector++ = *sources++
                        +
                        (
                        get_capacity_pcm (thermal_grid, dimensions, layer, row, column,temperatures)
                        /
                        step_time)
                        *
                        *temperatures++ ;
            }
            else
            {
              *vector++ = *sources++
                        +
                        (
                        get_capacity (thermal_grid, dimensions, layer, row, column)
                        /
                        step_time
                        )
                        *
                        *temperatures++ ;
            }
        } // FOR_EVERY_COLUMN
    } // FOR_EVERY_ROW
} // FOR_EVERY_LAYER
```

The function *get_capacity_pcm* is based on the preexisting *get_capacity* and assigns a volumetric heat capacity relevant to the temperature of the cell. For this reason, one might observe, that this new function is also passed the temperature value of the cell as a parameter. Regarding the conditional used, we assume that the melting point (the value that is immediately available for a material) of the PCM is located directly in the middle of (T1,T2), according to the chart presented in 3.9. To clarify further, Figure 3.10 was created.

As illustrated in the figure, in the phase change region, a ramp-like function is employed. If the melting temperature of the material, the melting duration and the maximum volumetric heat capacity are known (C_{tr}), then the function is fully defined. We consider that the heat capacities for the solid and liquid phases have the same value and are already known (the PCM material has already been declared in the stack description file). With these in mind, we can see that when a cell enters the phase change region, it is assigned a heat capacity value that increases linearly. This is continued for exactly one third of the melting duration. Afterwards, for the second third of the melting duration, the cell is assigned a steady maximum capacity value that is calculated by use of Equation (4). We can easily see that for the last third the capacity decreases linearly to reach the liquid heat capacity at temperature T_2 where we consider the material to be entirely liquid. Because the increase in heat capacity results in a decrease in temperature rate of change, for each cell undergoing phase change, the heat capacity follows the ramp-like function in Figure 3.10 quite thoroughly. At this point, we must examine the way in which the simulator will be made aware of all the extra variables.

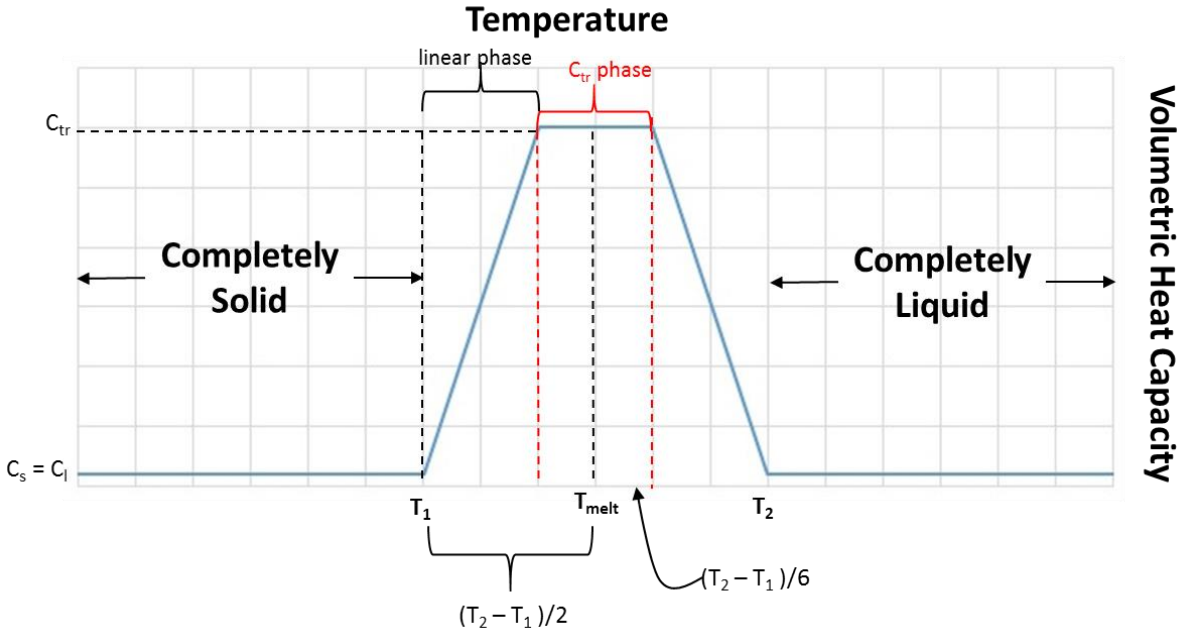


Figure 3.10: Ramp-function used to assign capacity values to PCM cells

If we intended to simulate a single material, with the values that define the ramp-function stable, hardcoding them directly to the program would be the way to go. However, in our

case, where a multitude of materials will be examined, changing values in the source code for every simulation hardly seems ideal. For this reason, we implemented the use of a PCM file (probably inspired by the stack description file) and created an appropriate function to parse it during the initialization phase. Sure enough, the values included in this file are the melting point, the maximum volumetric heat capacity and the melting duration (T_2 - T_1). Arguably, a similar entry could be included in the already existing stack description file coupled with the appropriate modification to the parser. However, this approach presented a number of difficulties that made our approach quite easier to develop and more time-effective.

Listing 3.4: Original for-loop filling the values of matrix **A**

```
FOR_EVERY_LAYER(layer,dimensions)
{
    FOR_EVERY_ROW (row_index, dimensions)
    {
        FOR_EVERY_COLUMN (column_index, dimensions)
        {
            tmp_matrix = add_solid_column

                (tmp_matrix, thermal_grid, analysis, dimensions,
                 lindex, row_index, column_index) ;

        } // FOR_EVERY_COLUMN
    } // FOR_EVERY_ROW
} // FOR_EVERY_LAYER
```

Seemingly, having also dealt with the problem of inputting the phase change material characteristics to the simulator, one would think that the additions required to properly model various configurations of PCMs are concluded. On the contrary, this is not the case. A simple test simulation would show the temperatures of the PCM cells to skyrocket when entering the phase change region in contrast with the expected behavior. If we take a look at Equation (3) again, we can see that the fact that was neglected is that matrix **A** is also dependent on the volumetric heat capacity of each cell. Consequently, recalculation of matrix **A** is also required.

Hopefully, 3D-ICE already provides functions to properly destroy the matrix and recreate it. Undoubtedly, this approach in not very performance-effective. Especially if we notice that only the diagonal elements of the matrix change. Nevertheless, in an effort to make our approach as non-invasive as possible, we opted to just destroy the matrix with the use of built-in functions, and repeat the creation process as was presented by the authors. The only change is that now, instead of calling the function *fill_system_matrix*, we call a modified version named *fill_system_matrix_pcm*.

The sole change in the new function, is that the latter contains a conditional that results in calling the *add_solid_column_pcm* function instead of *add_solid_column*. Those in turn, use the functions *get_capacity_pcm* and *get_capacity* which were described earlier. The result of this

chain of calls is that cells satisfying the conditional, get their capacitance from our appropriate function while all the others, get the value they would have under normal operation. The segment of code that was in effect and the one that took its place are presented in listings 3.4 and 3.5, respectively.

Listing 3.5: Modified code to accommodate dynamic volumetric heat capacity allocation

```
FOR_EVERY_LAYER(layer,dimensions)
{

  FOR_EVERY_ROW (row_index, dimensions)
  {
      FOR_EVERY_COLUMN (column_index, dimensions)
      {
          if (
              (strcmp(thermal_grid->material_id[lindex],"PCM")==0)
           && (*temperatures>=(thermal_grid->PCM_Melting_Point-(thermal_grid->PCM_Melting_Duration/2)))
           && (*temperatures<=(thermal_grid->PCM_Melting_Point+(thermal_grid->PCM_Melting_Duration/2)))
             )
          {
            tmp_matrix=add_solid_column_pcm

              (tmp_matrix, thermal_grid, analysis, dimensions,
               lindex, row_index, column_index,temperatures);
              temperatures++ ;
          }
          else
          {
            tmp_matrix = add_solid_column

              (tmp_matrix, thermal_grid, analysis, dimensions,
               lindex, row_index, column_index) ;
               temperatures++ ;
          }

      } // FOR_EVERY_COLUMN
  } // FOR_EVERY_ROW
} //FOR_EVERY_LAYER
```

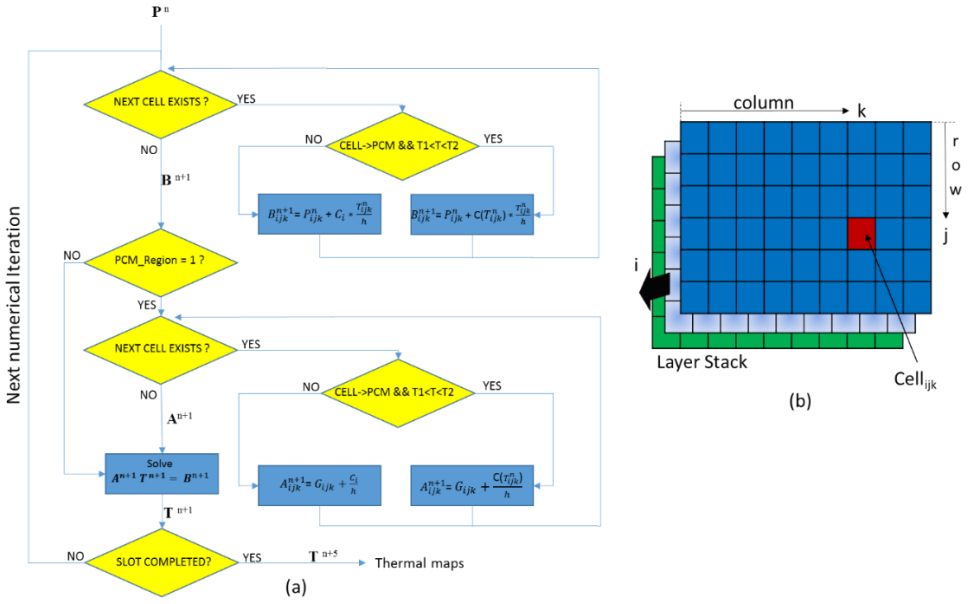


Figure 3.11: Execution flow for PCM-Enabled Thermal Simulation (a) Schematic Diagram, (b) Thermal Grid Representation

Furthermore, in order to avoid recomputing matrix **A** even when no PCM function is present, we implemented the use of the variable *PCM_Region*. This variable is checked before matrix **A** is destroyed and recalculated. *PCM_Region* is set to 1 when any cell satisfies the conditional we formed, that is, enters the designated phase change temperature region and belongs to the PCM layer. Obviously, even a single cell entering the phase change area requires the recalculation of **A**. In every other case, the value of *PCM_Region* is 0.

The fact that a generally big matrix is reinitialized and filled from scratch, even when only a handful of values need to be actually changed is not lost to us. However, as mentioned earlier, optimization considerations are outside the scope of this thesis. The prevailing fact in this case was that our approach was efficient with respect to development time and resulted to tolerable simulation time overhead.

As a side note, let us clarify that the segments of code presented were the most crucial parts, the most tightly related with the concept of the apparent heat capacity method. Other various and more mainstream sections enabling the specific implementation (e.g. variable declarations, new structures, value computations etc.) are not detailed in this text. The overall process described in this section is illustrated schematically in Figure 3.11.

3.3.3 Non-Uniform PCM Modelling

Through the use of our simulation framework, and after testing various phase change material configurations, the concept of using heterogeneous PCMs was found attractive. Specifically, the use of materials that differ only in the melting temperature but are part of the same layer was considered. In order to simulate such a configuration we should examine the role of the melting temperature in the program code. The first and foremost application of this variable, is in the conditional we use to determine whether we are in the phase change region. The second, is to determine the center of the ramp-function we use to assign capacity values. If we assume that all other variables characterizing the PCMs (phase change duration, max heat capacity) are common for all materials (for the sake of simplicity), and we substitute the *PCM_Melting_Point* with a likely named matrix, accessed by the rows and columns of each cell (variables already available at that point), we will have achieved the desired result without any further modification.

For that matter, if we expand the concept to the *PCM_Melting_Duration* and *PCM_Max_Heat_Capacity*, that is, we substitute the use of these variables with matrixes storing the value of each cell in the corresponding place (e.g. *PCM_Melting_Duration[row][column]*), we can easily simulate materials with different properties in general. The only matter left to be addressed, is to decide on a way to input the corresponding matrices. Of course it would be possible to hardcode the matrices directly and modify them when deemed necessary. Again, this approach does not seem very attractive if one intends to experiment with different distributions in the layer. The alternative approach is to formulate the matrixes in text files and create a function to parse them. Conceptually, it would be also possible to create files similar to the floorplan files, where instead of components, materials will be named, with their dimensions, positions and

characteristics. This option is the most complete but should be accompanied by a utility program to facilitate the creation of the floorplan files and a more sophisticated parser. In our case, the second option was applied.

3.4 Independent Tools - Scripts

3.4.1 MATLAB Scripts

Using MATLAB [24], we produced two basic scripts in order to visualize thermal and power maps. In truth, one script was used to visualize thermal maps produced by 3D-ICE which was later modified in order to be able to implement the same function for power maps. By visualization, we mean a color allocation to the different values included in the maps, upon which we superimposed a graphical representation of the appropriate floorplan. Each of the two scripts also uses a utility file. All of the above operations are handled completely by the scripts along with the archiving of the produced images to appropriate directories. The use of these scripts is facilitated by *Post_Simulation_Tools.py*, a small Python script copied in each archive folder which can be executed at any time after a simulation is stored. The use of these scripts could be integrated in the framework flow without great modifications. However, this integration was avoided due to the overheads in simulation time and memory they incurred, coupled with the fact that visual representation of all thermal and power maps was not always necessary.

3.4.2 ArchFP

ArchFP [25] was modified and used in order to produce the floorplans that were required in our simulations. The area values that were needed as input in ArchFP were gathered from McPAT. This tool was also used in order to produce the graphical representation of each floorplan. The output of ArchFP is not in suitable format for 3D-ICE. To create floorplan files in the appropriate format for the thermal simulator, a Python script was used that converted the files from ArchFP. This process, in cases where much variability in architectures and chip floorplans is present, could be integrated in the framework flow. In our case though, such integration was not deemed necessary.

3.4.3 Independent Python Scripts

In this section, the bulk of the scripts that were used outside the flow of the simulation framework are listed:

- **ArchFP_to_3DICE.py:** A small script that produces floorplan files suitable for 3D-ICE by processing the floorplans generated from ArchFP.
- **Create_Core_Aware_Floorplans.py:** Produces a text file with the rows and columns corresponding to the boundaries of each core in the floorplan. This text file is parsed by 3D-ICE in order to keep track of per-core temperature metrics.
- **Post_Simulation_Tools.py:** Offers four basic function-tools that can be used on demand after a simulation. The first two functions, *Visualize_Thermal_Maps* and *Visualize_Power_Maps*, run the corresponding MATLAB scripts as were described earlier. The function *Temperature_Metrics_Over_Time*, gathers and lists the maximum, average and minimum temperature of each layer as a function of time. The last function in this script, *Power_Trace_Per_Component*, creates a text file that contains the power values over time for each simulation component. All the functions employed in *Post_Simulation_Tools* are defined in *Post_Simulation_Utils.py*.

3.5 Framework Usage

In order to use the framework described in this chapter, the sole requirement is to edit *Simulation_Queue*, fill the lists with the variables of the simulations to be run, and execute the script. Prior to issuing any simulations, *Python_Server* must be started once. If no developing is to be conducted, *Python_Server* can be started with the nohup flag, thus disassociating the process from a terminal window, and later sent to run in the background. Obviously, a machine containing all the necessary files, tools and scripts that compose the framework is a prerequisite. The variables available in *Simulation_Queue* are grouped in three categories: Common Variables, Simulation Variables and Thermal Variables. In detail:

Common Variables:

- **Interval_NS:** The time interval used both in SniperSim and 3D-ICE in nanoseconds.
- **Heat_Distribution:** A choice regarding the interpretation of the power values outputted by McPAT. As mentioned earlier in this chapter, the interconnection between the power and thermal simulation is materialized by a Python script that parses the output values from McPAT and passes them on to 3D-ICE in an appropriate format. The discrete components that are listed in all the floorplans we used for the Gainestown Nehalem architecture and for which we obtained power values, are demonstrated in Figure 3.12. In addition to the components shown in the figure, we also obtained power values for the NoC of each core and the L3 shared across cores. In the scope of this thesis, four heat distribution schemes were used and are available for all chips using the same components as those described in our floorplans : Homogeneous, Discrete, Coarse and Fine-Grained. The Homogeneous scheme sums all power values and evenly allocates the result to all components. The Discrete scheme assigns to each component the power value calculated from the output of McPAT. The Fine-Grained scheme teams the power values to blocks containing more than one components. The blocks defined are: L2, Floating Point ALUS, Integer ALUS,

Complex ALUS, Register Files, Instruction Scheduler, Renaming Unit, Memory Management, Instruction Fetch Unit, Load Store and L3. The components corresponding to each block are seen in Figure 3.12. The NoC is integrated to the L2 block. The Coarse – Grained scheme teams up all ALUS with the Register Files and the Instruction Scheduler. In the Instruction Fetch Unit the Renaming and Memory Management Units are also added. All other components are the same as before. The blocks that result are Execution Unit, Instruction Fetch Unit, Load Store, L2, and L3.

- **Input_From_File:** A list with two values. Setting the first to NO will issue a normal execution as described in this chapter and ignore the second value. Setting the first value to YES, signals to the framework to conduct only thermal simulation, ignoring the performance and power simulation tools and using a text file with a power trace as input. In this case, the second value of the list must contain the path to the power trace text file.
- **Number_of_Simulations:** The number of requested simulations. This number is used to repeat simulation variables that are common for simulations rather than having to type them as many times.
- **Run_Mask:** A list with a numeric value for each simulation requested. Allowed values are 0 and 1. Simulations marked with 0 in the Run_Mask are ignored and not run. This variable facilitates the creation of more general patterns in the simulation queues allowing any unnecessary simulation to be skipped.

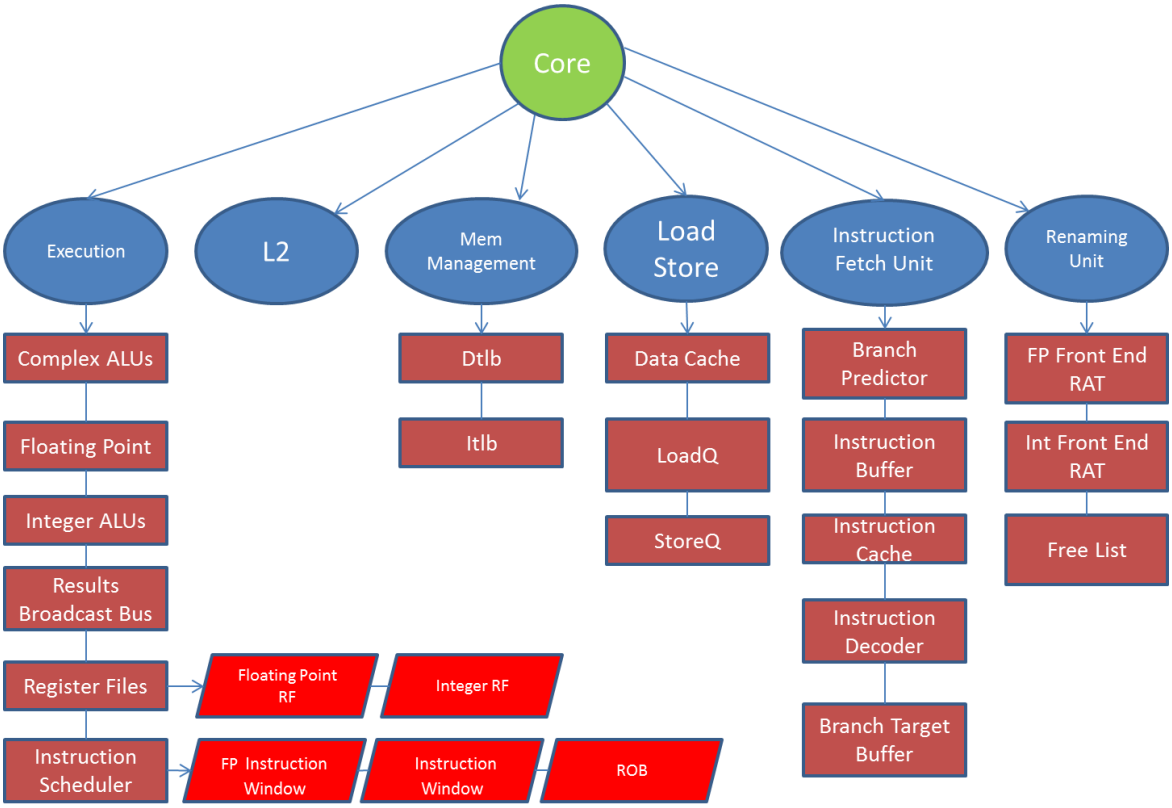


Figure 3.12: Core Discretization used in floorplan creation and power simulatio

Simulation Variables:

- **Benchmarks:** A list with the benchmarks for each simulation. Multiple benchmarks for the same simulation are a single string in the list with the requested benchmarks separated by commas. Benchmarks are listed in the format supported by SniperSim, meaning, *suite-app-worksize-threads*.
- **Platform_Cores:** A list containing the number of cores in the chip for each simulation.
- **Architectures:** A list containing the architecture of each platform. In our work, only Gainestown was used but the variable has been exported to this level to facilitate the framework expansion if necessary.
- **Scripts:** List containing the scripts to be invoked by SniperSim. The scripts are followed by any input variables separated by semicolons. In our case, inputs to *Sniper_Simulation_Control* were Interval_NS, Input_From_File[0] and Heat_Distribution.

Thermal Variables:

- **Chip_Size:** A list of lists with the inner lists containing two values describing the size of the chip for each simulation (length, width) in μm.
- **Cell_Size:** The cell discretization to be used by 3D-ICE (length, width) in μm in a similar format with the previous variable.
- **Sink_Area:** The area of the heatsink in μm^2. If no heatsink is specified later, this value is ignored.
- **Simulation_Step:** The duration of each simulation step within a time slot for 3D-ICE. The time slot corresponds to the simulation interval while the number of steps that can fit in a slot represent the number of iterations of the numerical method.
- **Initial_Temperature:** The initial temperature of the cells in the chip.
- **Heatsink:** Available options for this list of variables are YES and NO indicating whether the chip in equipped with a heatsink or is connected directly to the ambient.
- **Layers_List:** A list of lists declaring the elements in the chip stack from top to bottom. An example: [‘SINK’,’SPREADER’,’PCM’,’DIE’,’PCB’].
- **Heights_List:** A list of lists, declaring the height of each element specified in the stack, respectively. Example: [‘2000’,’1000’,’200’,’100’,’10’]
- **Materials_List:** Same as before, declaring the material of each element from top to bottom. Example : [‘COPPER’,’COPPER’,’LAURIC_ACID’,’SILICON’,’BEOL’]. A set of materials and their properties are hardcoded in *STK_Utils.py*. If new materials are intended to be used, then an entry to this script must be added. Adding materials to this script does not necessitate using them always. One might declare a great range of materials and use them on demand. That is the reason why we did not export this variable for immediate editing.
- **Melting_Points:** Contains the melting point for each PCM used in the simulations. If no PCM is specified in the chip stack, the corresponding value in this list is ignored.

Apart from all the variables specified in the list, a full system simulation also involves some files still. As was described earlier, a floorplan file is necessary as input to 3D-ICE. To create the

floorplans for Nehalem-like configurations we used ArchFP and transformed the output to a suitable format. However, the fact that the operation of ArchFP is an interactive trial and error process, until the specified layout is achieved, precluded us from automating the floorplan creation and integrating it to the framework. For this reason, pre-created floorplans for Nehalem-like architectures with cores ranging from 4 to 128 cores are fed directly to 3D-ICE. If another architecture is to be used, the user must see to providing suitable floorplans. Along with these, if a visual representation (image) is included in the same directory, the framework will be able to visualize thermal and power maps with respect to this image, if so requested. A core file is associated with each floorplan we created designating the rows and columns occupied by each core in the layout. This file is used by 3D-ICE to provide per core temperature statistics and is unique to the architecture and discretization parameters. In order to utilize this function for another architecture, a suitable core file must be included. If no such file is present, the function and the corresponding statistics are not employed.

In summary, in order to use the simulation framework, the necessary actions can be divided conceptually into two groups. The first group, dealing with prerequisites, is the one linked with files generally regarded as stable, involving framework, floorplan and floorplan associated files (visual representation, core boundaries). The initiation of the Python server is also categorized in this first group. In the second group, we regard the definition of all variables relative to the simulations we want to run. The distinguishing factor between the two, lies in the fact that the first group is characterized by little or no repetition, whereas the second group is replicated for each batch of simulations.

A cumulative view of the inputs at each stage of the framework is shown in Figure 3.13. Inputs that are stable during the execution of a simulation are marked in black, in contrast with inputs that are different for each interval marked in red. All blocks designated with green color are materialized with Python scripts whereas the blocks in blue are independent tools.

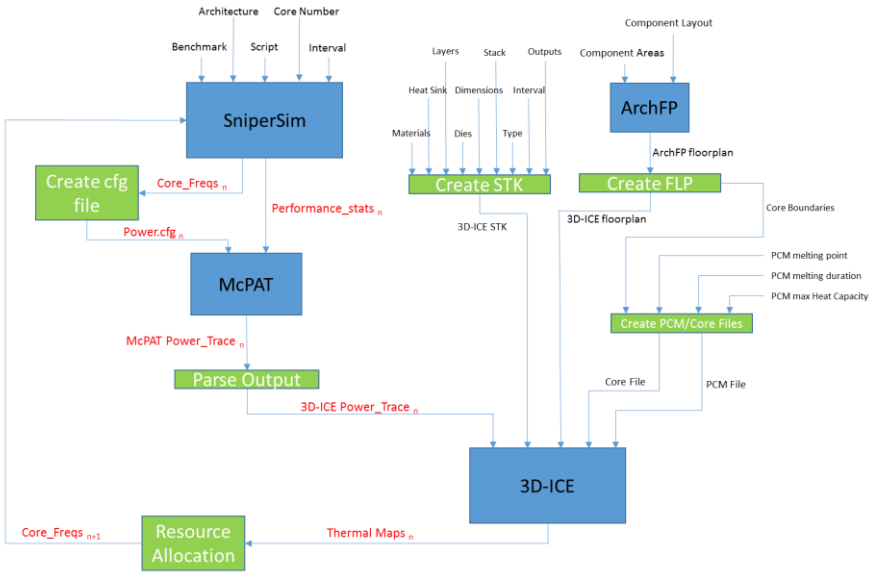


Figure 3.13: Input-wise overview of Simulation Framework

CHAPTER 4

4.1 Theoretical Background

This chapter is devoted to showcasing the importance in thermal modelling, of interface materials between chip components and of heat distribution along the core die. Notably, this analysis is only an example of the usage that the full system simulation framework we described in Chapter 3 can be put to. In order to better understand the variables involved and the reasons that led to this analysis, a brief theoretical background will be provided in this section.

Thermal Interface Materials, commonly known as TIMs, are increasing in importance in nowadays ever more power hungry and small chips. Heat generated by semiconductor devices must be removed to the ambient to maintain the junction temperature in the silicon die within safe operating limits. This heat removal process, often involves conduction from a package surface to a heat spreader that can more efficiently transfer the heat to the environment. The spreader in turn, might be connected to a passive or active heat sink to further boost the cooling capabilities of the system. Regardless of the case, the spreader has to be carefully joined to the package and/or sink surface to minimize the thermal resistance of this newly formed joint.

Attaching a heat spreader to a semiconductor package surface requires that two commercial grade surfaces be brought into intimate contact. These surfaces are usually characterized by a microscopic roughness superimposed on a macroscopic non-planarity that can give surfaces a concave, convex or twisted shape. When two such surfaces are joined, contact occurs only at the high points. The low points form air-filled voids. Typical contact area can consist of more than 90 percent air voids, which represents a significant resistance to heat flow.

Thermally conductive materials are used to eliminate these interstitial air gaps from the interface by conforming to the rough and uneven mating surfaces. This is illustrated graphically in Figure 4.1. Because TIMs have greater thermal conductivity than the air they replace, the resistance across the joint actually decreases, counterintuitively to the fact that we are adding an excess layer (and in theory, excess thermal resistance) between the conducting surfaces. In truth, what happens is that we replace a poor conducting material, air, with a better one, the selected TIM. Thus, the resulting thermal resistance across the two surfaces diminishes. It stands to reason that the same effect is also transferred to the joint temperature. Reducing the resistance from the die to the ambient generally results in cooler thermal profiles.

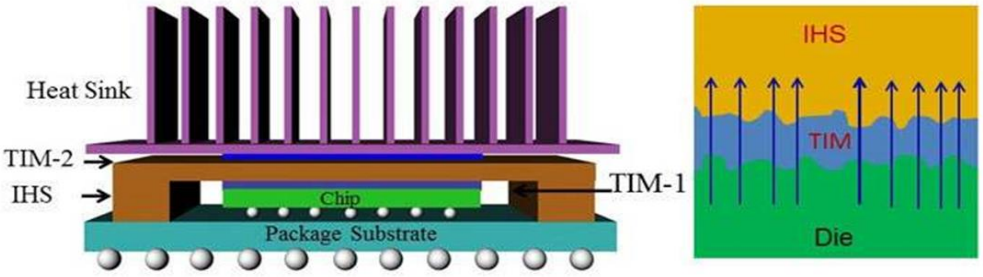


Figure 4.1: (a) Chip components arrangement in a Flip Chip LGA Package
(b) Graphical illustration of TIM usage between die and IHS

Regarding heat distribution across the core die, it has been shown in the literature that the temperature distribution in a microprocessor is architecture and/or workload dependent. In [26], it has been demonstrated that placing sensors at the hottest locations determined by one application can cause large temperature errors in temperature readings for other applications. The authors go on to explain that certain different applications incur different power and thus, temperature profiles across core components. As a result, localized heating is application dependent. Since the research in [26] was conducted on a certain microprocessor, in order to generalize the previous conclusions, the authors outline cases in the literature where diverse results were reached in attempts to identify the hottest regions of other microprocessors. Regardless of this diversity in results, one might notice that the object of all related research lies in identifying patterns that link different applications and process variations to specific components demonstrating higher temperature profiles. In any case, the fact that the temperature across the die components is not uniform, with large differences reported, is undisputable.

This heterogeneity of temperature profiles has a noteworthy impact on the overall thermal behavior of chips, due to the fact that thermal conduction is greatly dependent on power density. To clarify further, a component of certain area, consuming stable power will result in quite a different temperature profile than the total of the processor die consuming the same power. While seemingly obvious, this fact is often neglected in thermal modelling schemes, where power is allocated evenly across the core components, coupled with an overestimation in power consumption to reflect worst case, full-utilization scenarios. This approach, while seemingly analogous or at least, safely producing a worst case scenario, will be seen to fail to capture accurately both transient effects across stack layers and power densities that create localized hotspots.

It is our belief that this uneven power and consequently, temperature allocation across the core die will be further exacerbated in chips intending to use frequency sprinting, due to even higher component consumption. In addition, the use of phase change materials, which is very popular among such configurations, is often characterized by non-uniform melting of the PCM layer. As reported in [9], failure to capture the phenomenon where parts of the phase change material are melting while other parts are still solid, can result in significant under or over-estimation of temperature.

While this fact is stated with different power consumption and heat production among different cores in mind, the non-uniform temperature allocation in the die of a single core can be seen from two different perspectives, closely linked to the particular concept. From a per-core perspective, the single die can been thought of, as a microscopic variation of a multicore chip demonstrating localized hotspots, thus effecting the conclusions presented in [9] in a smaller area. To elaborate, not accounting for non-uniform melting in the PCM across a multicore chip is analogous to not accounting for non-uniform melting in the PCM across a single core die, albeit in smaller proportions. Certainly, the penalty in simulation accuracy will be smaller in a per-core basis.

However, more careful consideration of the problem will soon reveal that this two phenomena are tightly coupled. In fact, a simple analysis will show that they work additively. Bearing in mind both that components in a single core and cores in a single multicore might exhibit higher utilization and power consumption than others, one can easily deduce that the combined effect of this two factors, leads to even more focused hotspots. Hence, the inaccuracies described in [9], are further exacerbated when assuming per-core uniform heat distribution.

To make matters worse, thermal interface materials, as briefly described previously, are almost always neglected in thermal modelling. On its own, this omission is an indirect assumption of ideal surface conformation between layers in the stack. Obviously, the resulting thermal model will be, in general, better able to conduct heat. More specifically, depending on the placement and characteristics of the TIMs neglected, the thermal model will be unable to model transient heat spikes correctly and simultaneously lead to underestimation of steady state temperatures.

In conjunction with all the preceding analysis, not modelling thermal interface materials is expected to lead to even greater irregularities in the thermal model. Truth be told, each and every one of the factors mentioned, has an impact in modelling accuracy on its own, but is also influenced by all the others. To summarize, we believe that uneven heat distribution across the die of a core leads to different temperature profiles than uniform distribution. This difference, will be more prominent in sprinting frequencies. In addition, coupled with the absence of TIMs in the thermal model, the resulting temperature map of the chip will be smoothed out. The same line of reasoning can also be followed backwards. Not modelling thermal interface materials leads to smoother profiles, which are smoothed out even further due to erroneously assuming homogeneous heat distribution across core components which is further apart from true conditions in sprinting cores. Both factors, heat distribution and thermal interface materials, seriously affect chips that intend to employ phase change materials and sprinting methodologies.

In order to validate this theory we just presented and demonstrate the impact of the factors involved, we formulated a series of simulations. In this chapter, we will demonstrate the different results in each scheme at increasing frequency levels, for some different workloads. The initial goal is to demonstrate the increasing effect of heat distribution and TIMs with increasing frequency levels and thus prove their importance in sprinting oriented research. The modelling approach and all related variables, will be presented in section 4.2.

4.2 Simulation Methodology

In order to test the theory we presented in section 4.1, we decided to conduct our experiments on a four core Nehalem Gainestown based chip. To better simulate larger power densities, consistent with current technology trends, we opted to use a 22nm process, extracting component sizes and chip areas from McPAT. McPAT reports a die size of 48,358 mm^2 which was modeled as a 8 mm x 8 mm square chip, assuming a 20% error, which was the average under evaluation reported in [18].

To simulate the above chip, SniperSim was used with an interval of 1ms. The benchmarks used to conduct the experiments are Blackscholes, Bodytrack and Streamcluster, all for a total of 4 threads and for the duration of 1000ms. All benchmarks used the large work size. Due to the fact that no dynamic control over performance simulation characteristics was conducted, to accelerate simulation times, we recorded the power trace of each benchmark using our framework, and then fed it directly to our thermal model through the options available from the Python interface. In cases where the input did not reach the desired levels, we iterated over the values of the parallel region. The reason we focused our analysis on the first 1000ms of each case, derives from the fact that the average power levels in the ROI of the above benchmarks is pretty much stable. As a result, since we wanted to focus on transient effects and temperature spikes, we felt that iterating over the same values would only serve to demonstrate the resulting steady state levels which we deemed not equally important. Nevertheless, a brief discussion of the results from a 10.000ms simulation are addressed in every section. Each benchmark was simulated for all cores working in 2190, 2390, 2660 and 2926MHz.

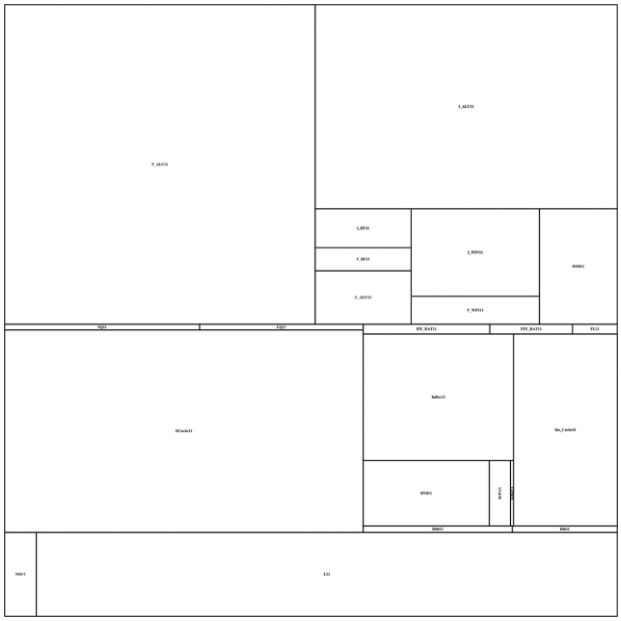


Figure 4.2: Single core floorplan

Regarding the floorplan and the components placement, we used ArchFP to produce the necessary files, as mentioned in previous sections, and we based the layout on information from [27]. The component areas were retrieved from McPAT as mentioned earlier. The resulting floorplan, in visual representation, can be seen in Figures 4.2 and 4.3.

Figure 4.3: Four core chip floorplan with L3

For our thermal model, we decided to use a variation of the chip with a heat spreader but no heat sink. This decision was influenced by the work presented in [10] where a similar configuration was used. We further selected to use a heat spreader that is the same size as the die. This creates an even more challenging power profile for the chip and accounts for the limited space characterizing mobile chips. It is of course noted, that heat spreaders need to be mounted on the PCB and need to have a bigger surface than the die to achieve their maximum potential and placement requirements. However, we selected to sacrifice a portion of lateral heat spreading to create an even more thermally challenged multicore and provide more options, later on, for possible PCM configurations.

As far as the chip stack was concerned, we used the model that can be seen in Figure 4.1, representing a flip chip land grid array, without the heat sink. A more practical representation, showing the actual stacks used in the thermal model, is shown in Figure 4.4. The materials of the heat spreader and of the die are, of course, copper and silicon respectively. The thermal interface material assumed was indium, which is reported by [28] to be one of the best choices available. Lastly, a small layer representing the underfill with parameters derived from [29] was also present. The parameters used for each of these materials are summarized in table 4.1.

In 3D-ICE, we used 100 μm as cell length and width and used 250 μm thick layers in the stack. In [19] it is reported that dimensions of a few hundred microns are sufficient for accuracy, hence, the selected values are both acceptable in terms of precision, and result in small simulation times. The layers that had smaller thicknesses were of course modelled accordingly. Transient analysis was conducted with a time slot of 1ms, the same used in SniperSim. The step was set to 0.2ms resulting in 5 steps, that is, 5 iterations of the numeric method, in each interval. The initial temperature of cells was set to 300 K. The top layer of the heat spreader was connected directly to the ambient and a value slightly worse than the one reported in [9] was used for the convection

resistance, to account for worse heat transfer conditions due to the fact that no heat sink is present. Sure enough, this value was adjusted to the heat transfer coefficient required as input for 3D-ICE. Furthermore, in each simulation, we outputted thermal maps for each layer along with the power map for the die. The maximum, average and minimum temperature of each layer through time was also recorded along with the maximum temperature corresponding to each core die.

Material	Thermal Conductivity	Volumetric Heat Capacity
	W/m*k	J/ m^3 * K
Copper	400	1.57*10^6
Indium	80	1.70*10^6
Silicon	130	1.62*10^6
Ceramic	1	2.10*10^6

Table 4.1: Material properties used in thermal simulations

In order to demonstrate the validity of the theory we formulated in 4.1, we used four different configurations for each benchmark at each frequency level. The first, denoted as A1, is the default chip stack we just described, modeled with fine-grained heat distribution. The second, is the same configuration without the TIM layers. The other two are identical but employ homogeneous power distribution among all core components. This is achieved with the same power trace by accordingly modifying the values for each interval, such that the power density is uniform across the core die and the same total power is dissipated in the chip. Let us note also, that 4 total threads for each benchmark does not mean that 4 threads are always active in the simulation. It only means that 4 threads will be spawned in total and are not necessarily active all the time. This is ideal in our case, since it provides variability both across the die of a single core, and across the cores of the chip. The configurations are numbered A1 through A4, in the order they were presented. The same information is also presented in Figure 4.4.

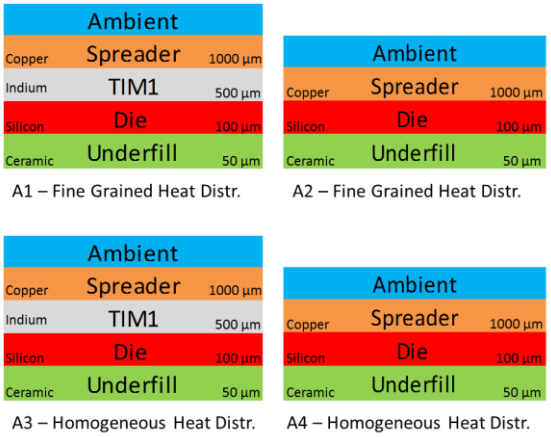


Figure 4.4: Thermal configurations used in the simulations

4.3 Results and Discussion

4.3.1 Blackscholes Simulations

We decided to first present some results for each benchmark separately, analyze them in some detail and then summarize our findings. This approach was chosen in order for the reader to be better aware of the individual characteristics of each configuration before being presented with overall statistics. As was already explained, Blackscholes was run with 4 total threads for 4 configurations (A1-A4) at 4 different stable frequencies. To begin with, let us examine the temperature traces, with respect to the maximum temperature observed in the silicon die, presented in Figure 4.5.

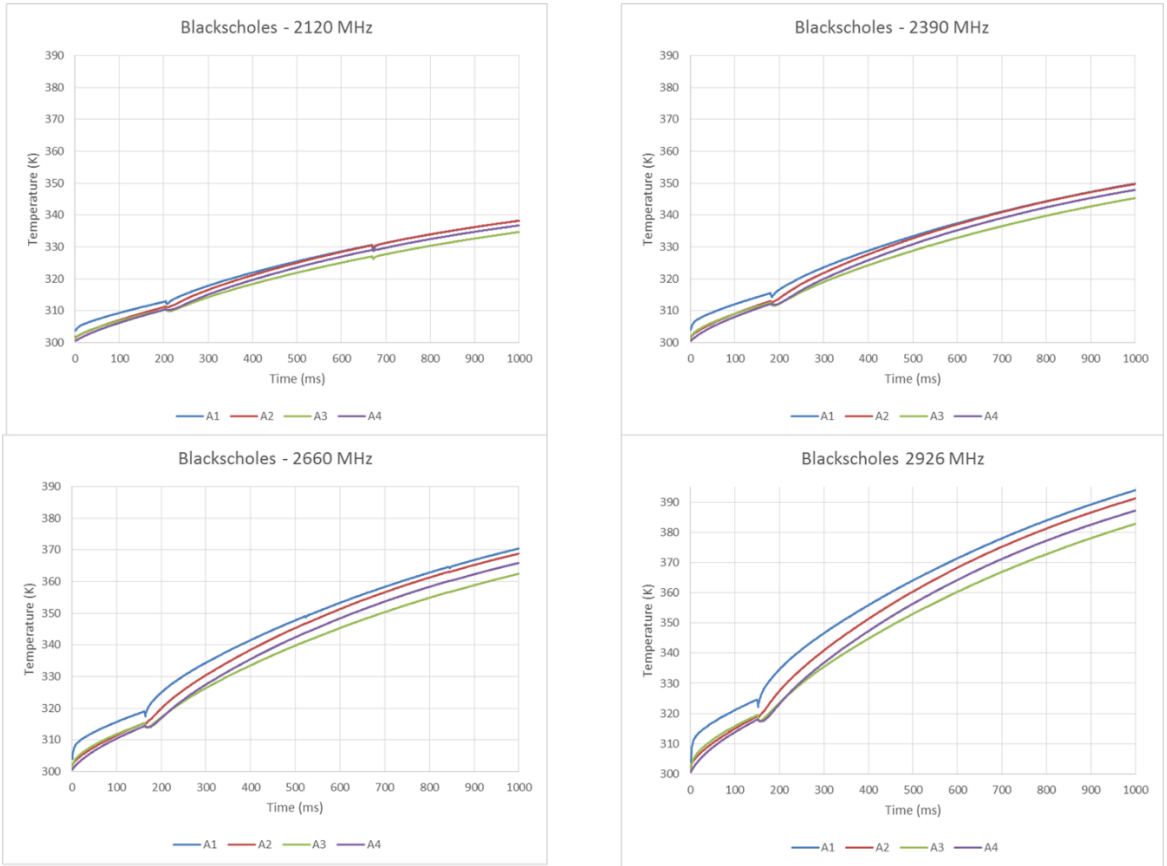


Figure 4.5: Thermal traces for Blackscholes at increasing frequency, for each configuration

What is readily observed from the charts, is that the slope of the curves increases with frequency. Put more simply, that the maximum temperature in the silicon increases as frequency

increases, which is quite expected. A more careful examination, will also reveal that as frequency increases, so does the gap between the temperature traces of each configuration.

This gap, one might notice, is more prominent at the beginning of each simulation. This happens because that time region corresponds to the serial phase of the benchmark where only one core is active. The operation of this single core, for this particular benchmark, is characterized by an increased power density in the center region of the core. The result is a tightly focused hot spot. For reference, Figure 4.6 shows the power traces for each core and the corresponding sum for the chip. The first core in the chip, or alternatively, Core 0, is moderately active in the beginning of the simulation and then reverts to idling. The utilization of all the other cores is pretty much even in the parallel region.

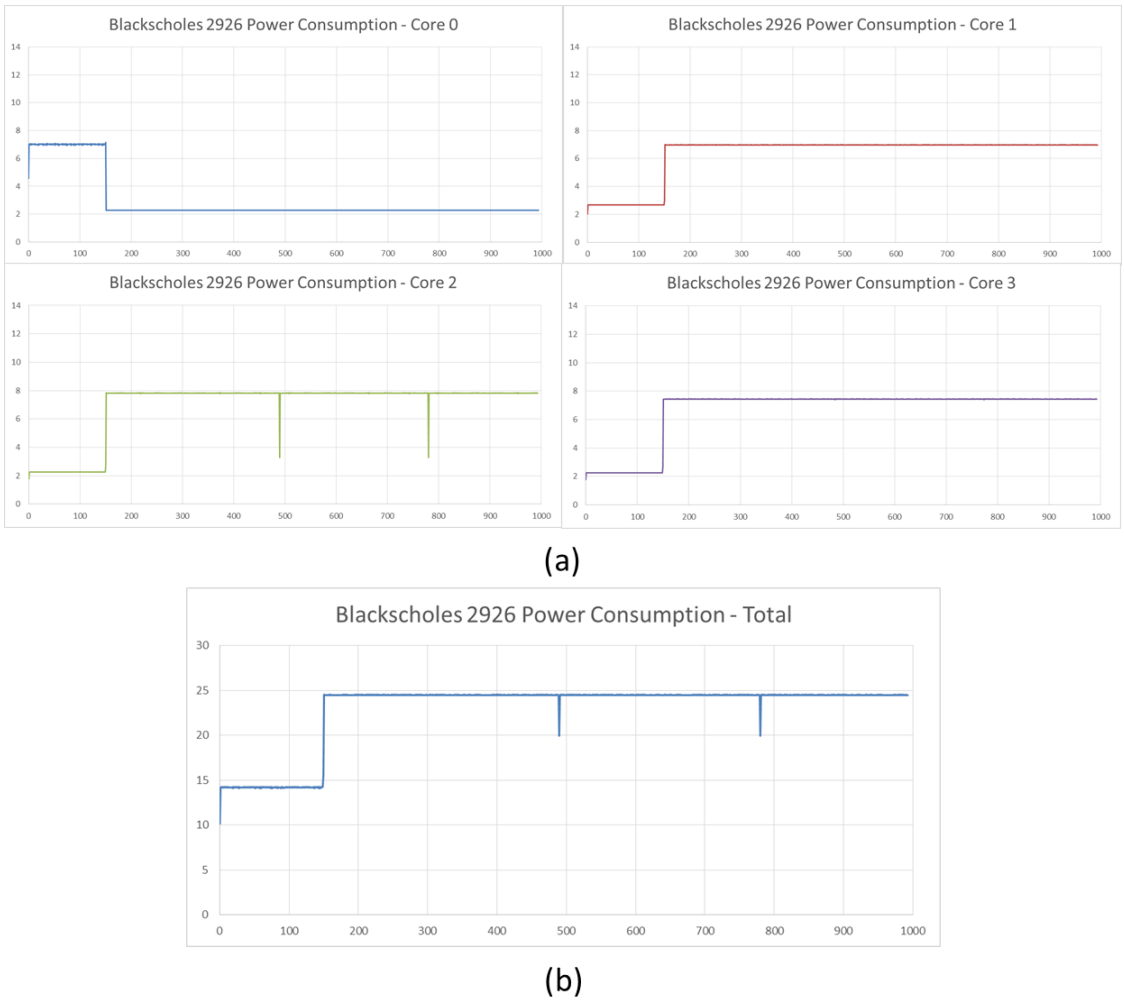


Figure 4.6: Per-core and total power traces for Blackscholes at 2926 MHz

In the serial phase, we can see from the diagrams that for configuration A2, even though fine grained heat distribution is used, the hot spot from the operation of one core, with some more power consuming components standing out, is very quickly smoothed out. The reason for this, is

that the die is modeled in direct contact with the copper heat spreader. The spreader, true to its name, by virtue of the very high thermal conductivity of copper, quickly conducts the heat concentration to the surrounding cells, resulting to a quite lower maximum temperature. Configuration A3, is seemingly closer to the temperature levels of A1. The reason being that even though homogeneous heat distribution is considered, the modelling of TIMs, captures the hot spot generated by the operation of a single core, while all the others remain dark. The result is that it leads to higher maximum temperatures than the other configurations at this point. Configuration A4, using homogeneous heat distribution without thermal interface materials, simulates the operation of a single core, equally active at all components that is easily spread out with respect to the generated heat.

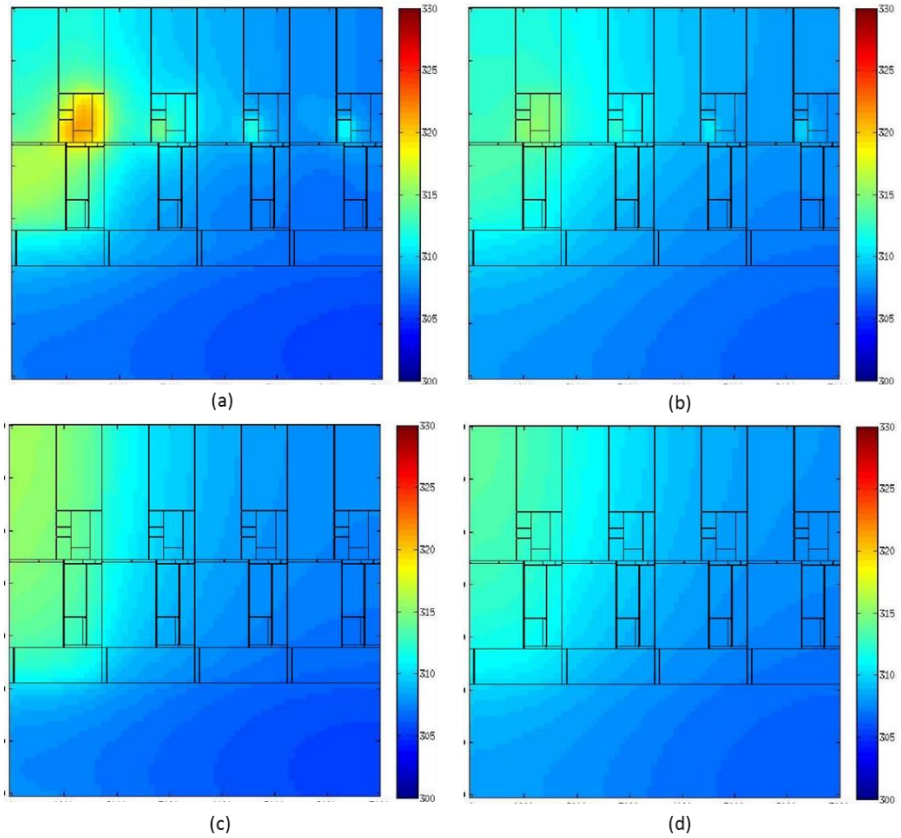


Figure 4.7: Thermal profiles for each configuration at t=100ms
(a) A1 – Accurate modelling of hot spot
(b) A2 – Accurate power model for hot spot, inaccurate thermal model
(c) A3 – Inaccurate power model for hot spot, accurate thermal model
(d) A4 - Inaccurate power and thermal model for hot spot.

It is worth noting that configurations 2, 3 and 4 do not demonstrate big variations in the resulting maximum temperature value. Apart from the fact that they all fail to simulate the localized hotspot correctly, they have certain characteristics that tend to mitigate their differences.

A3 models a single homogeneous core with slow spreading of heat to neighboring cells. Since the core selected was the first of the four presented in the floorplan, that means that A3 dissipates heat slowly towards the second core and the common interface with the L3. A2, by spreading the heat generated in the hot spot seen in 4.7- (a) very efficiently, has a tendency to converge to the profile created by A3. By virtue of the centered position of the hotspot, heat is dissipated at a rapid pace, towards the rest of the first core (thus converging to the A3 initial modelling profile), and towards the second core that is close to the hotspot (part of the evolution of the A3 modelling profile). Configuration A4, is a combination of the two and stands farther apart, but is kept a bit reigned from the fact that the faster pace at which it is dissipating heat is only utilized towards two directions (Core 1 and L3), since the other faces of Core 0 are at the edge of the chip boundaries.

To better understand the results described in each case, a thermal snapshot of the chip for each configuration is presented in Figure 4.7. The thermal snapshot is taken at 100ms elapsed time for the 2926 MHz case. In this figure, we can see the hot spot we previously described in (a), the same hot spot, but greatly smoothed out in (b), the uniform operation of one core with small heat spreading in (c), and the uniform operation of one core with rapid heat spreading in (d). From this figure, another important fact is unveiled. Apart from the absolute difference in maximum temperature reported in each configuration, we can easily see that different thermal profiles are created in each case. This fact will be shown to have a serious impact in PCM enabled configurations.

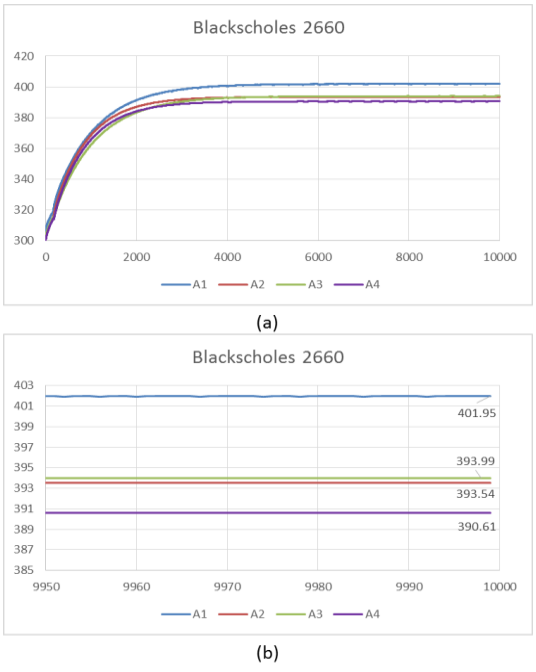


Figure 4.8: Temperature trace for Blackscholes iterated over the parallel region at 2660 MHz

(a) Temperature trace overview

(b) Last 50ms of simulation demonstrating steady state values

Returning to Figure 4.5, we can see that after entering the parallel region at about 150ms, in each configuration the max temperature starts converging to a steady state value. In that case, where three cores are active, the temperature differences are smaller as heat is less focused and less variable. An interesting fact is that configuration A2 tends to close the gap towards A1 on account of the faster heating of components due to the high conductivity of copper coupled with the smaller heat capacity in the system. In other words, reverting to the electrical analogy, A2 has a smaller RC factor meaning that with steady, constant value input, the voltage (temperature) will rise at a greater pace. Simultaneously, the final steady state temperature of A2 will actually be lower since the thermal resistance from the die to the ambient is smaller. To recap, A2 will converge faster to a smaller steady state temperature than A1. This will be showcased later on.

A similar behavior is also observed between A3 and A4. In this case, because these configurations were closer in temperature, A4 has time to elevate to greater values than A3, but later on, will start converging to smaller temperatures for the reasons explained previously. For all cases, the same simulation carried on for 10s is shown in Figure 4.8. We can easily see in 4.8 (b) that as expected, A2 converges to a smaller value than A1, and A4 to a smaller value than A3, even though, we can see in 4.5 that A4 is rising at a faster pace initially.

In addition, for comparison purposes, the thermal profile for each configuration is shown after 700ms in Figure 4.9. Note that in this case the color reference has changed, with the cooler colors corresponding to 350 kelvin degrees. Naturally, the temperatures reported in 4.8 are prohibiting for the function of the chip. This fact is neglected in this phase since the object of our interest are the differences in the thermal models. The maximum temperature for the silicon die is commonly regarded as 100 Celsius degrees, the equivalent of 373.15 in kelvin. In our work, we used a more conservative limit of 370 kelvin as the value at which we report overheat of die components.

In order to quantify the differences in thermal simulations outlined so far, as well as present an alternative point of view, we created a set of statistics from the thermal traces we logged. These statistics are demonstrated in Figure 4.10. The error values listed are derived by comparing the thermal trace for the maximum temperature of A1 with every one of the other configurations. As expected, A4 is the worst case with respect to the maximum error observed. A3, follows pretty close in all frequencies while A2 generally exhibits the smaller error. This outcome was expected due to the previous analysis where A2 was seen to be better able to follow, even from a certain distance, the thermal trace of A1. This tends to point that heat distribution is a more important factor in the case we are examining and that disregarding thermal interface materials in cases where homogeneous heat distribution is assumed, results in small irregularities in comparison. Regarding the average error values, A2 exhibits accordingly the lowest values while A3 results in a bigger average error due to the phenomenon we described earlier in this section. Of course, as we have seen, configurations 3 and 4 do not result in much different thermal profiles, with the difference being that in small time windows, 4 exhibits higher temperatures. As a result, the better average performance is largely artificial.

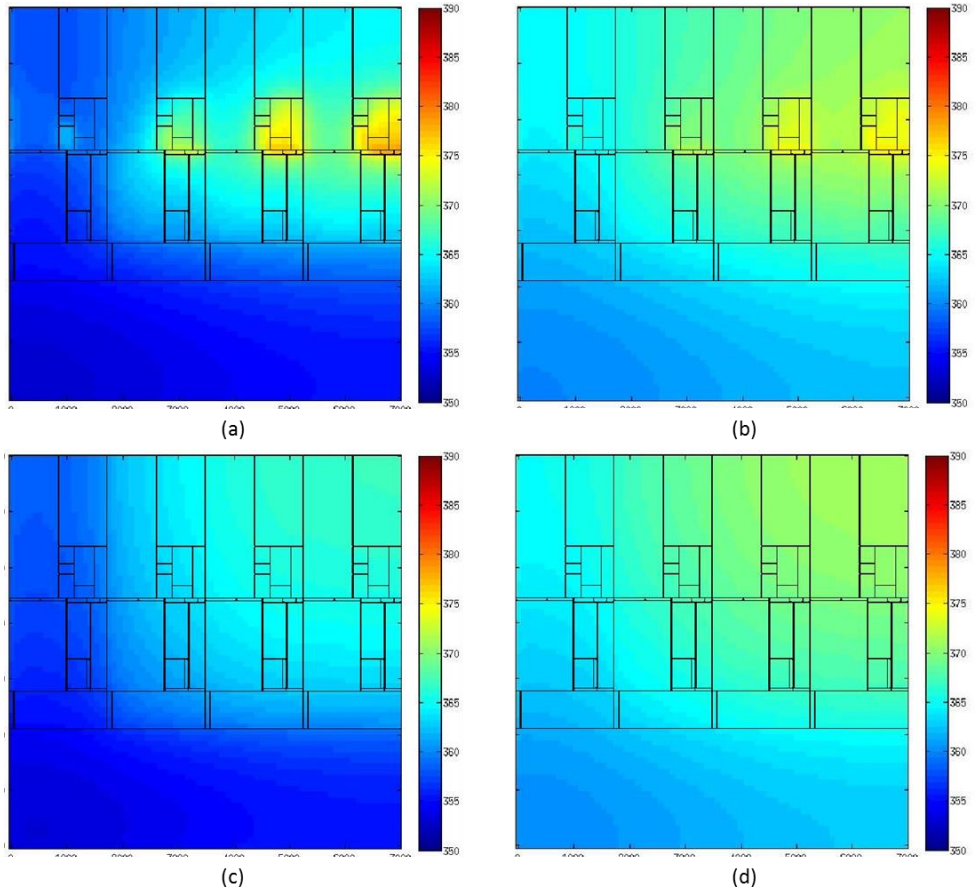


Figure 4.9: Thermal profiles for each configuration A1 through A4 at t=700ms

Figure 4.10: Maximum and average error values for Blackscholes temperature traces

Regarding the effect of frequency on the inaccuracies that surface, each of the configurations exhibited an increase in maximum error of 200% on average, and an increase in average error of 300% on average, all in the span of less than 1GHz increment in frequency. Regardless of the individual numbers, the upward trend in maximum and mean error values is indisputable and quite intense. The specific values represented in Figure 4.10 are summarized in table 4.2.

While all the facts presented so far seem quite interesting, another point of view might be taken into account, a point very often neglected. Earlier in this text, we mentioned the usual limit for the temperature in the die. Temperature measured in the center of the die is denoted $T_{junction}$. The corresponding limit for temperature at the center of the top level of the heat spreader, commonly known as T_{case}, is usually about 20 degrees lower. That means that chips with maximum $T_{junction}$ at 100 degrees Celsius, use 80 degrees Celsius as the limit for $T_{case.}$ Essentially, this means that there is an expected 15 degrees temperature difference between core and case temperatures, at loads that exceed cooling capabilities (that means more than the expected full load). In our work, in tune with the described trends and the choice for $T_{junction\_max}$, the maximum value for T_{case} was set at 350 Kelvin which is a little bit lower than the 353.15 kelvin degrees equivalent of 80 C°.

MHz\ C°	Maximum temperature error			Average temperature error		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	2.80	3.61	3.87	0.82	3.34	2.15
2390	3.72	4.64	5.00	1.16	4.28	2.85
2660	5.21	8.10	8.15	2.83	7.28	5.46
2926	7.61	11.28	11.60	4.30	10.22	7.91

MHz\ C°	Maximum temperature error %			Average temperature error %		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	0.92	1.14	1.27	0.26	1.03	0.67
2390	1.21	1.44	1.63	0.37	1.29	0.87
2660	1.65	2.46	2.51	0.84	2.10	1.60
2926	2.31	3.34	3.49	1.23	2.82	2.23

Table 4.2: Comparison of A1 against all other configurations – error values

Another interesting weakness of said thermal models occurs in this case. In Figure 4.11, it can be seen that in any case examined, excluding A1, the temperature of the top spreader layer is very close to that of the die. Especially, in the 2926 MHz case, the only one that violates temperature limits clearly, at the moment of the first violation of $T_{junction}$, in A1, the temperature of T_{case} is 353 Kelvin. This number approximates quite well the anticipated behavior, in contrast with the corresponding 365, 363, and 368 Kelvin for T_{case} in the other configurations, respectively. Even if we assume that the 20 degr2ees gap usually assumed between T_{case} and $T_{junction}$ is an approximate towards ensuring that the die never reaches critical temperatures, meaning that the

actual difference might actually be smaller, the deviations presented in the other configurations are too large and imply further inaccuracies.

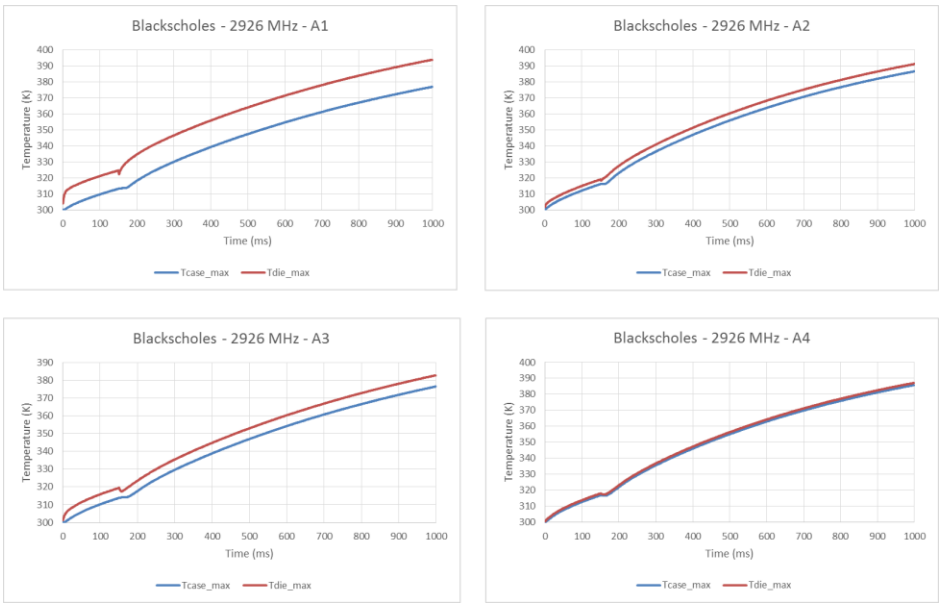


Figure 4.11: Thermal trace for $T_{junction}$ and T_{case} for each configuration at 2926 MHz

Although we used the temperature of the die as the trigger for our later exploration, as will be presented later, the distribution of heat in the vertical direction of the chip stack, is paramount when the use of PCMs is considered. As an example, assume that the optimal melting temperature for a PCM to be placed on top of the heat spreader is intended to be determined. We can see that methods 2, 3 and 4 would probably result in erroneously different results on account of modelling hotter top-most layers. The same will hold true for an exploration regarding the optimal position of a phase change material in the chip stack.

Lastly, let us underline that in the most intense frequency scenario, the time of the first temperature violation is reported at approximately 600ms, while, in configuration 4, the same first violation, is reported more than 150ms later. This underestimation, might be the most critical for systems intending to employ sprinting and phase change materials, since temperature violations trigger control actions and are used to determine the performance of methodologies and materials.

4.3.2 Bodytrack Simulations

In accordance with expected results and what was presented in the previous section, from the thermal traces for Bodytrack, shown in Figure 4.12, we can easily identify the upward trend in maximum die temperature, rising in gradient as frequency rises. Similarly, with every frequency step up, the gap, or more accurately, gaps between the thermal trace for each configuration, widen perceptibly. In contrast with Blackscholes, Bodytrack is characterized by fluctuations in the thermal trace.

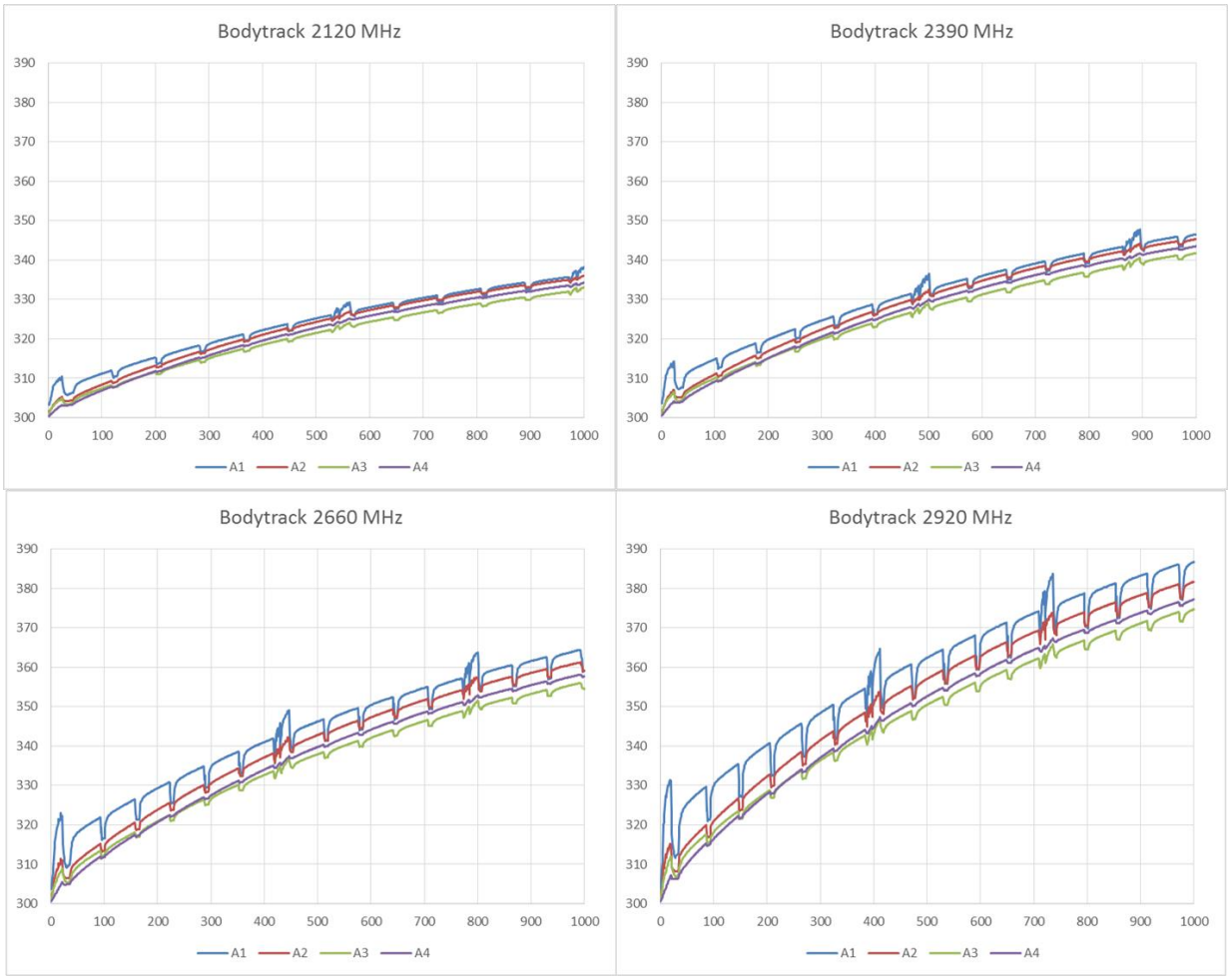
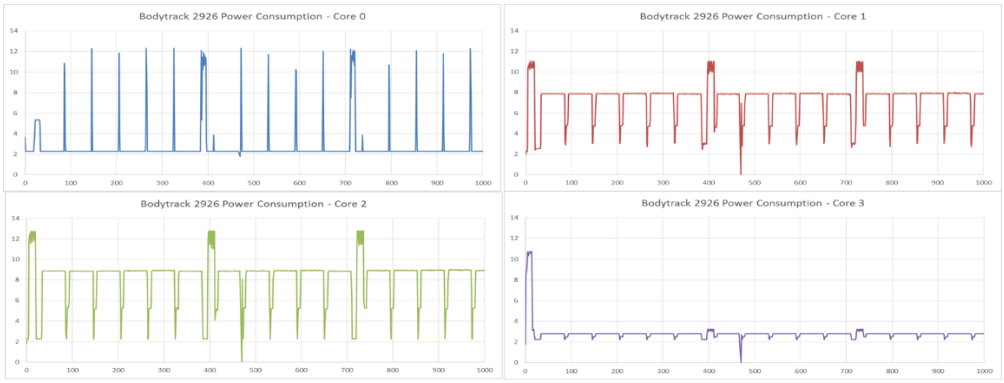
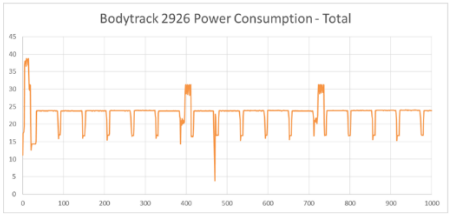


Figure 4.12: Thermal traces for Bodytrack at increasing frequency, for each configuration

To better understand this behavior, Figure 4.13 was created, presenting the power traces for each core, along with the power trace for the sum of the chip, for Bodytrack, with the cores operating at 2926MHz. It is readily seen that Core 3 is for the most part idling, while Core 0 is punctuated by short bursts of activity. Cores 1 and 2, have similar power traces in form, both with each other, and with the sum. An interesting fact to observe, is that the thermal trace of the maximum temperature of the chip, follows the output expected from an RC circuit, inputted with a voltage similar in form to the total power trace, superimposed on a curve representing the RC response to the average value of the power consumption of the whole chip.



(a)



(b)

Figure 4.13: Per-core and total Power traces for Bodytrack 2926 MHz

By way of a better explanation, les us divide the power trace for the total chip in two different functions of time, a steady value equal to the average power, and a transient value equal to the power trace shown in 4.13 (b) minus the steady value. The response of an RC circuit to the steady value, would be something similar to 4.14 (a). The shape of the transient value and the shape of the corresponding RC response, would roughly be what we can see in 4.14 (b). The sum of the steady and transient functions would be our power trace. The sum of the respective responses, would be approximately the sum of 4.14(a) with the response in 4.14 (b), which, would have a similar shape with our actual thermal trace.

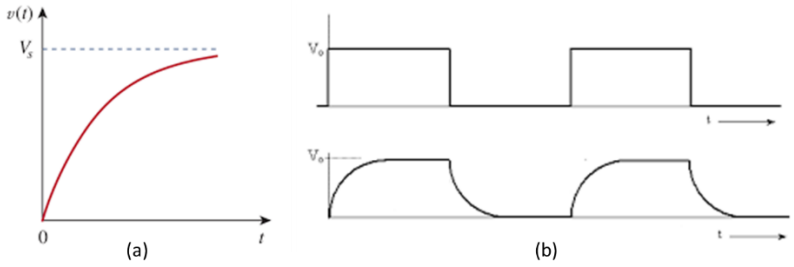


Figure 4.14: Example RC responses
(a) RC response to steady voltage input
(b) RC response to step input

Of course, the known analogy of electrical and thermal phenomena is taken into account. However, the fact that the overall maximum temperature of the chip follows closely the response of the total power through time, even though this power is the result of highly uneven distribution

among the cores, resulting in uneven thermal profiles, is intriguing. Nevertheless, it should also be noted that this behavior is very faint, in configurations 2 to 4.

The analysis regarding the steady state values and rate of change of temperature that was presented in the previous section, is also in effect here. Specifically, we can see that the curve corresponding to the average power consumption conforms to the previously specified behavior. This is not surprising since the characteristics involved are specific to each configuration. Exactly like in Blackscholes, A2 converges faster than A1 to a lower steady state value and A4 converges faster than A3 to a lower steady state value. These findings and the corresponding steady state values can be seen in Figure 4.15.

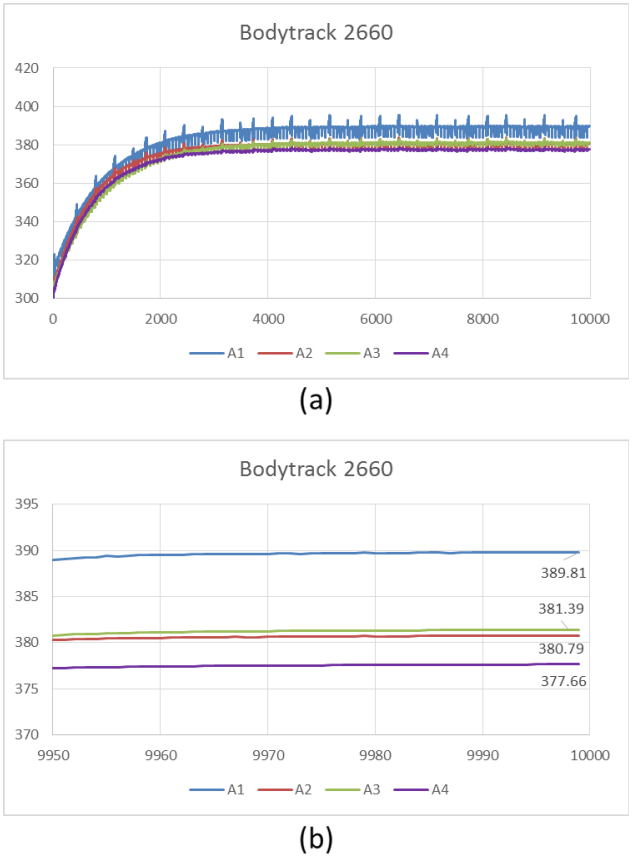


Figure 4.15: Temperature trace for Bodytrack iterated over the parallel region at 2660 MHz

(a) Temperature trace overview

(b) Last 50ms of simulation demonstrating steady state values

In order to observe the difference between the thermal profiles that each configuration generates, we created Figure 4.16. The images in the figure are thermal snapshots for each setup, taken every 100ms of time, up to 700ms. The more power consuming cores 2 and 3 are seen to be translated to hotter cores in configuration A1, as expected. In A2, the same result can also be noticed, even though, heat is much more spread out, and the hyperactive cores (with respect to the

others) are only slightly hotter. The outline of the active cores in A3 is distinct, and the same behavior that was demonstrated in A1, is demonstrated here, at core instead of component level. This of course, leads to smoother profiles because power is spread out and less focused. A4 demonstrates a scenario which only vaguely corresponds to the mismatched operation of only two cores.

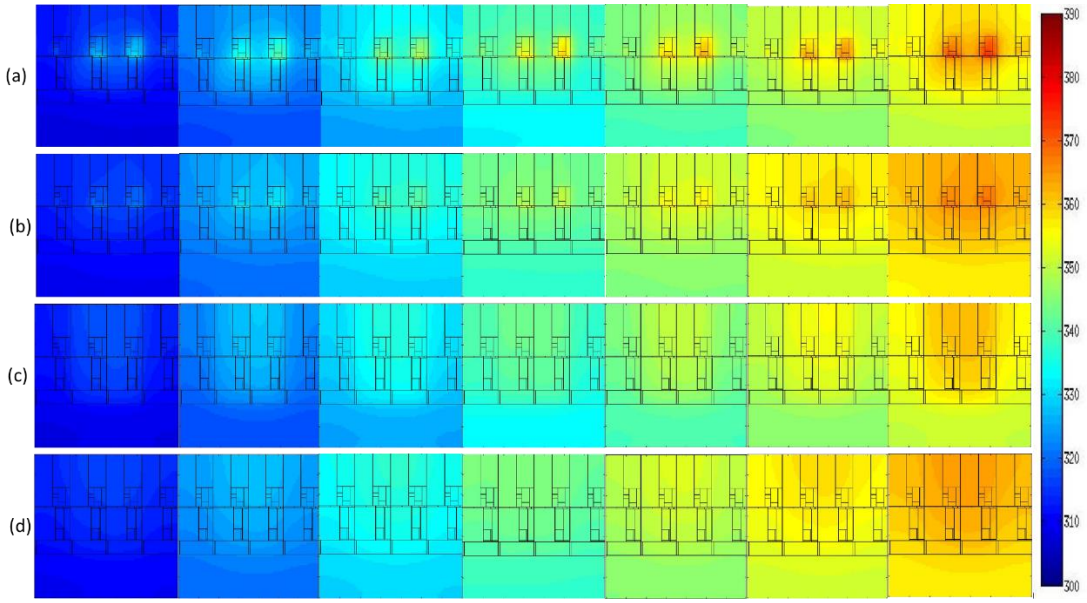


Figure 4.16: Thermal snapshots every 100ms for Bodytrack at 2926 MHz
(a) – (d) Configurations A1 to A4 respectively

In the overall results from Bodytrack, even before we examine the specific numbers, we can conclude that heat distribution is again more influencing. The scenario presented for configuration A2 is seen to be closer to A1 than any of the others.

The statistics that are derived from our simulations and are presented in Figure 4.17, and the corresponding specific values in Table 4.3, corroborate the remarks we made from observing the thermal profiles. Specifically, configuration A2 exhibits the smallest error both in maximum and average temperature, at all frequencies. A3 is next in magnitude of maximum temperature error, but shows the largest values for average metrics. This behavior is an artifact produced transiently from the higher upward slope of A4, in the time window we examine. As we can see in Figure 4.15, A4 diverges more than A3 later on in the simulation for a long period of time, leading to an overall larger average temperature error.

Let it be noted, that if we derive the same statistics for 10s of simulation instead of 1, the values that result are without exception the same or larger. However, we felt that that for smaller time windows, where the responses are still far from steady state values and the majority of temperatures reported are within silicon operating limits, the metrics better reflect transient operations that characterize abrupt core activation and/or frequency boosting in sprinting scenarios.

Figure 4.17: Maximum and average error values for Bodytrack temperature traces

In this benchmark, the average increase in maximum temperature error, from 2120 to 2926 MHz, is slightly larger at approximately 240%, whereas, the same metric for the average temperature error is slightly smaller at roughly 280%. At the same time, while it is not easily observed at this point, let us note that all error values are larger for Bodytrack, at corresponding frequencies and configurations. This is to be expected, since the overall nature of this benchmark is much more variable. This variability is the behavior that thermal models, neglecting thermal interface materials and/or uneven heat distribution across core dies, have difficulty simulating accurately.

MHz\ C°	Maximum temperature error			Average temperature error		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	5.13	5.57	7.21	1.26	3.63	2.65
2390	7.22	8.35	10.19	1.89	4.67	3.60
2660	11.48	14.87	18.15	3.83	8.05	6.81
2926	16.33	20.32	24.93	5.73	11.37	9.94

MHz\ C°	Maximum temperature error %			Average temperature error %		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	1.65	1.80	2.33	0.40	1.12	0.83
2390	2.30	2.53	3.24	0.58	1.41	1.10
2660	3.55	4.60	5.62	1.13	2.34	2.00
2926	4.95	6.16	7.55	1.62	3.17	2.80

Table 4.3: Comparison of A1 against all other configurations – error values

In brief summary, this weakness in modelling incurs maximum temperature errors that might trigger control actions at the wrong time, thus distorting all simulation results from that point forward. In addition, the different thermal profiles affect phase change materials reported performance. This is not yet quantified, but will be analyzed later on. Lastly, inaccuracies in vertical heat spreading, across stack layers, also affect simulated PCM performance, depending on the placement. An examination of Figure 4.16 shows the differences in temperature traces of $T_{junction}$ and T_{case} for each configuration. At the time instant that the die registers 370 K, the case temperature is 351 K for A1. The other configurations report 363, 363 and 368 K respectively at the same time instant.

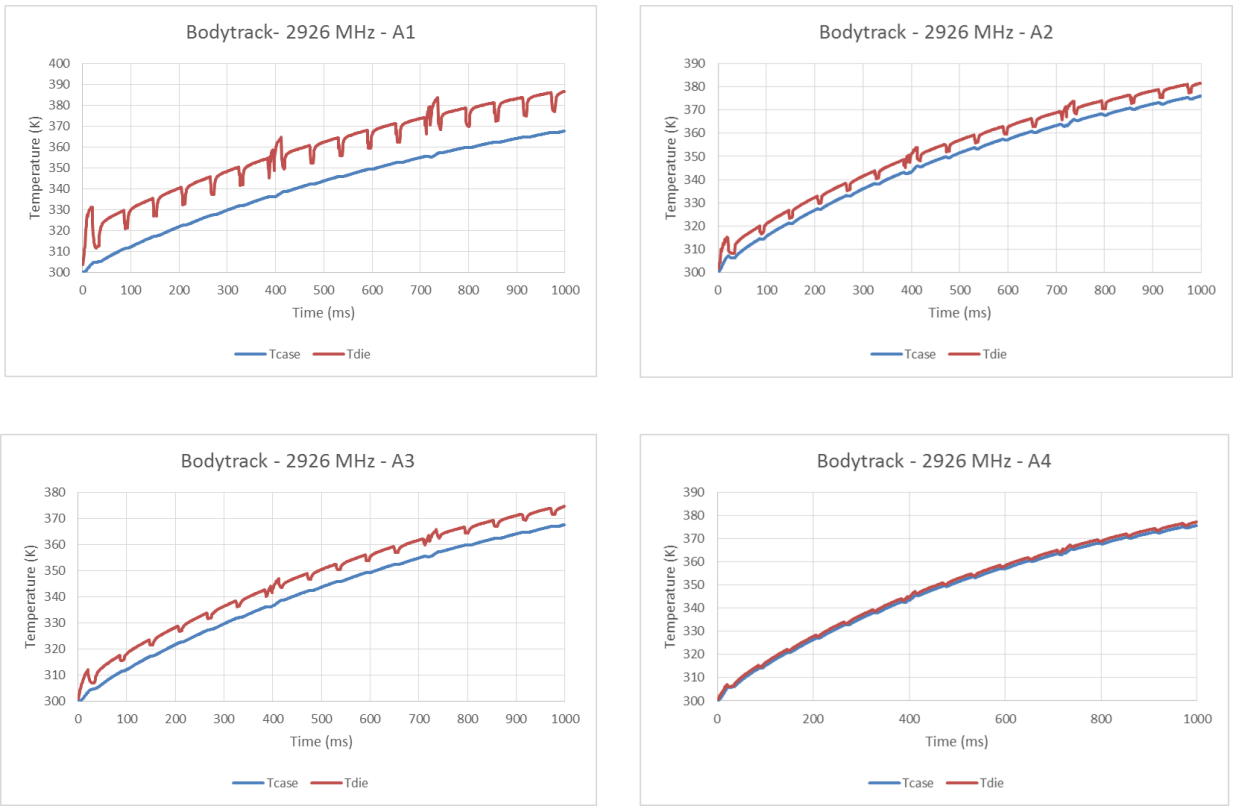
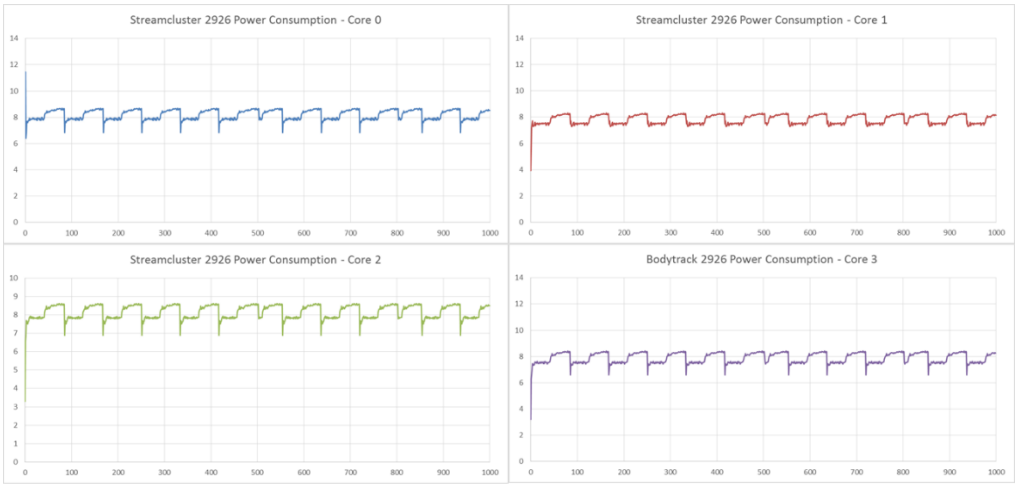


Figure 4.16: Thermal trace for $T_{junction}$ and T_{case} for each configuration ta 2926 MHz

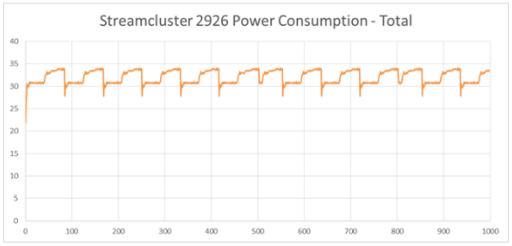
In the most intense frequency scenario, the time of the first temperature violation is reported at 626ms in A1. For the other configurations, the corresponding numbers are 716, 878 and 814ms respectively.

4.3.3 Streamcluster Simulations

For the case of Streamcluster, we used 4 active threads in the parallel region, corresponding to 5 total threads spawned. This choice, along with the specific benchmark, were combined in an effort to favor the models using homogeneous heat distribution and no thermal interface materials. Streamcluster is characterized by a very small serial region and approximately even power allocation among cores. This allocation along with the total power consumption can be seen in Figure 4.17.



(a)



(b)

Figure 4.17: Per-core and total power traces for Streamcluster at 2926 MHz

Using 4 active threads in the parallel region creates a quite uniform profile, which we expect will mitigate the spreading inaccuracies of the models, and result in smaller errors. In addition, the total power consumed by the benchmark fluctuates only slightly, removing abrupt heat spikes that are seen to be more difficult to model. With all these parameters in mind, we expect to see smaller errors on all accounts between the thermal models and more similar thermal profiles. This choice of parameters, intends to explore a case where a degree of uniformity is actually present in the multicore.

From Figure 4.18, we can see that the thermal traces for each configuration seem to be more uniform. Of course, with increasing frequency their differences start becoming more prominent and a small fluctuation of temperature, resulting from a corresponding small fluctuation in power consumption, starts becoming more intense. However, it should be observed that the reference scale for these charts is by necessity different. Because more threads were used in this case, the benchmark is consuming more power and, as a result, leads to higher temperatures in the same frequencies. The difference in scale means that errors in maximum temperature, tend to seem smaller in the charts presented.

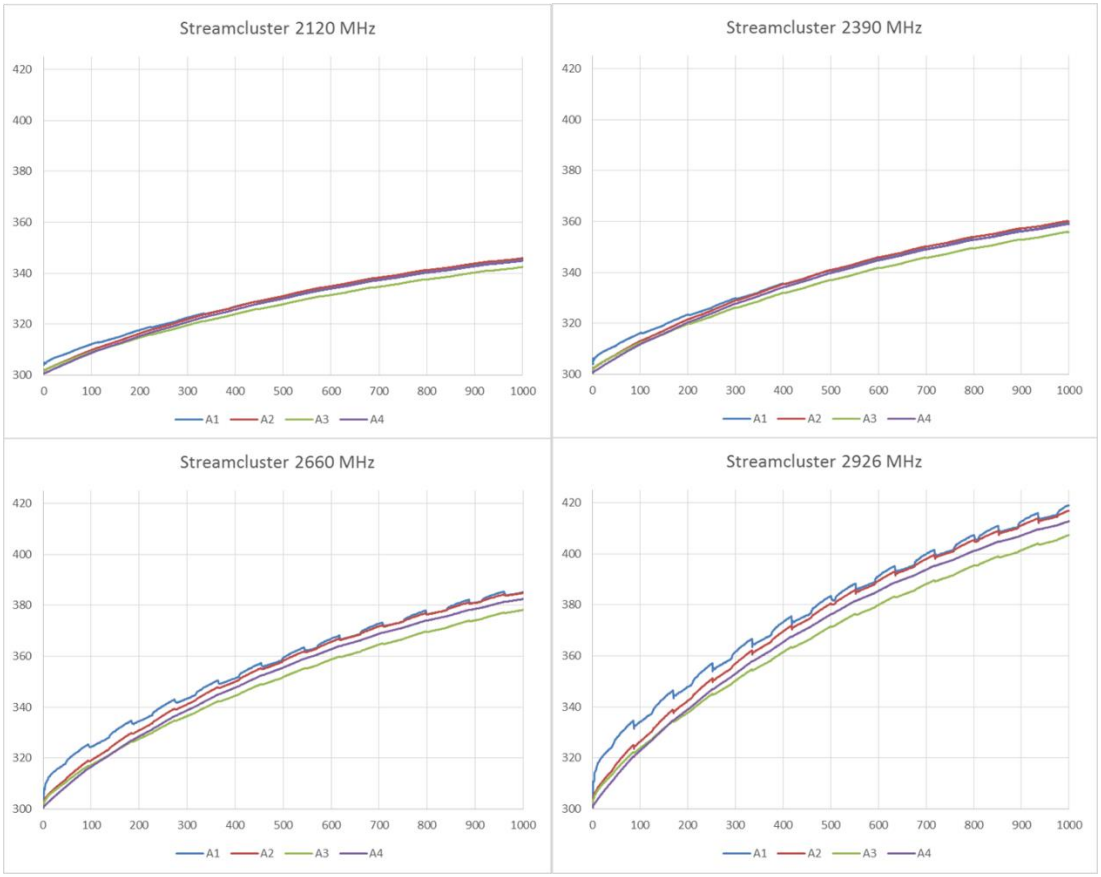


Figure 4.18: Thermal traces for Streamcluster at increasing frequency, for each configuration

The parameters that are specific to each configuration, meaning rate of convergence to steady state values (alternatively, rate of increase in temperature or RC factor) and the steady state values themselves, influence the thermal traces for this case according to previously detailed tendencies. The analysis we presented in 4.3.1 regarding transient behavior and expected steady state values is also in effect here. Nevertheless, said behavior is not equally visible in these thermal traces, on account of the different scale and smaller divergence between the configurations.

In order to get a better perspective, Figure 4.19 presents the simulations at 2660 MHz conducted for 10s. Regarding what we have seen so far, the curves in the figure are in accordance

with the expected results. Even though it is not clearly obvious, A4 and A2 advance in temperature at a faster pace, but later on settle to lower values than A3 and A1 respectively.

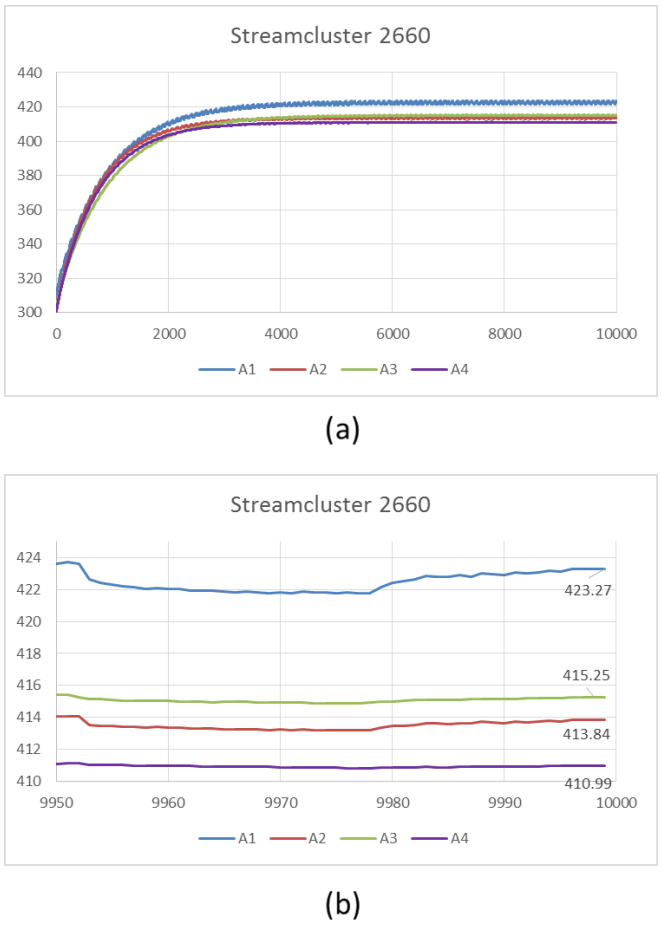


Figure 4.19: Temperature trace for Streamcluster iterated over the parallel region at 2660 MHz

(a) Temperature trace overview

(b) Last 50ms of simulation demonstrating steady state values

However, as was emphasized earlier, while statistics with regard to specific metrics are important in unveiling useful and maybe critical information, they are not adequate to provide the full image. For this reason, Figure 4.20 was created, with thermal snapshots from each configuration, taken at 50ms intervals up to 300ms. As was previously stated, thermal profiles, in a two dimensional, and a vertical sense, are important factors in configurations intending to use phase change materials. The intervals were chosen to demonstrate chip behavior closer to actual operating temperatures.

Even though configuration A1 still has a tendency to stand out, on account of the hotspots properly modelled in the center regions of each core, in this case, the differences between the thermal profiles are less intense. It can be seen that A2 is quite closer to A1 in this instance.

This results from the fact that all components are equally active, and dissipating heat, rendering the absence of TIMs (a factor that greatly influences heat spreading) less consequential. Regardless, even with this better conformation between thermal profiles, configurations A4 and A2, which do not model interface materials, are seen to lead to a substantial inaccuracy in the temperature of the L3. In a scenario where a phase change material is interposed between the die and the rest of the chip stack, this cooler part of the L3 might quite easily correspond to a portion with latency in melting. Naturally, failure to capture this behavior could quite easily lead to significant errors in determining the overall performance of the system.

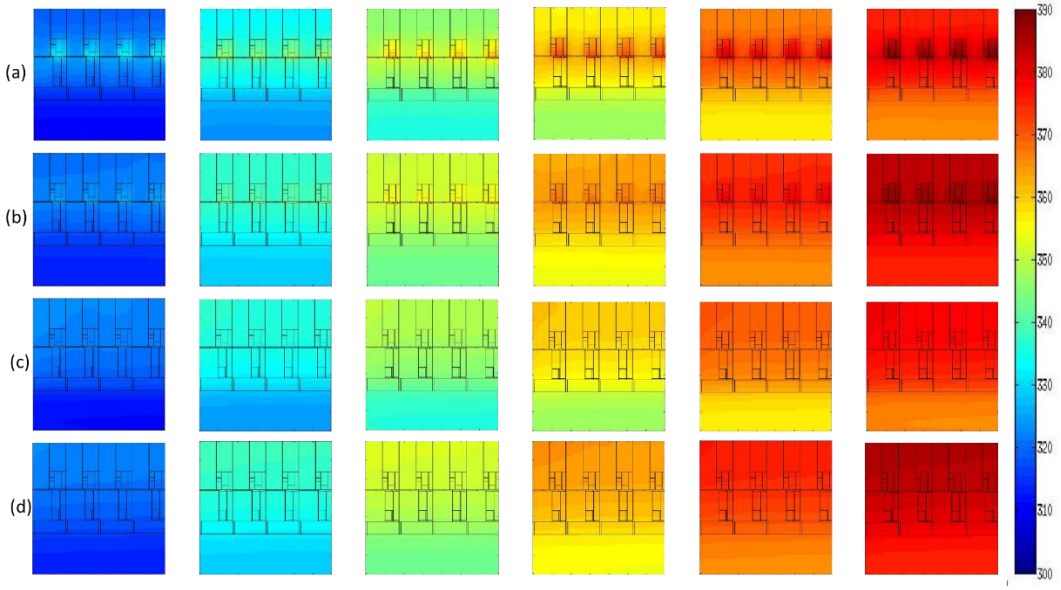


Figure 4.20: Thermal snapshots every 100ms for Streamcluster at 2926 MHz (b) – (d) Configurations A1 to A4 respectively

With regard to Figure 4.21, we can see the same overall trend presented in the previous sections. Even though Streamcluster incurs smaller errors than other benchmarks, a greater sensitivity to frequency is observed. The average increase of maximum error ascends to approximately 260% while the same metric for the average error registers an unexpected 350%, approximately. This result is counterintuitive to both our expectations and the thermal profiles we have already examined. This fact only serves to corroborate our claim that no metric on its own, is fully determinative of the thermal simulation differences. The specific values for the error metrics computed are listed in Table 4.4.

MHz\ C°	Maximum temperature error			Average temperature error		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	3.03	3.12	4.30	0.88	2.89	1.27
2390	4.01	4.06	5.57	1.14	3.58	1.55
2660	6.89	8.52	10.10	2.13	7.52	4.78
2926	10.07	12.45	14.80	3.45	10.90	7.27

MHz\ C°	Maximum temperature error %			Average temperature error %		
Frequency	A1-A2	A1-A3	A1-A4	A1-A2	A1-A3	A1-A4
2120	0.99	1.02	1.40	0.27	0.88	0.40
2390	1.30	1.33	1.81	0.35	1.06	0.48
2660	2.19	2.63	3.18	0.63	2.12	1.38
2926	3.08	3.75	4.51	0.97	2.90	1.99

Table 4.4: Comparison of A1 against all other configurations – error values

Figure 4.21: Maximum and average error values for Streamcluster temperature traces

Finally, our last but not least approach is demonstrated in Figure 4.22. Our simulation model consistently retains the expected temperature difference between case and die, despite the full and uniform utilization of the chip. Specifically, at 380ms, when the temperature of the die reaches 370 K, the reported temperature for the case is 351.5 K. In contrast, the reported temperatures for the other models are 364, 362 and 368 K respectively. Obviously, these temperatures correspond to the time instants when each model reports die temperature to have reached 370 kelvin degrees.

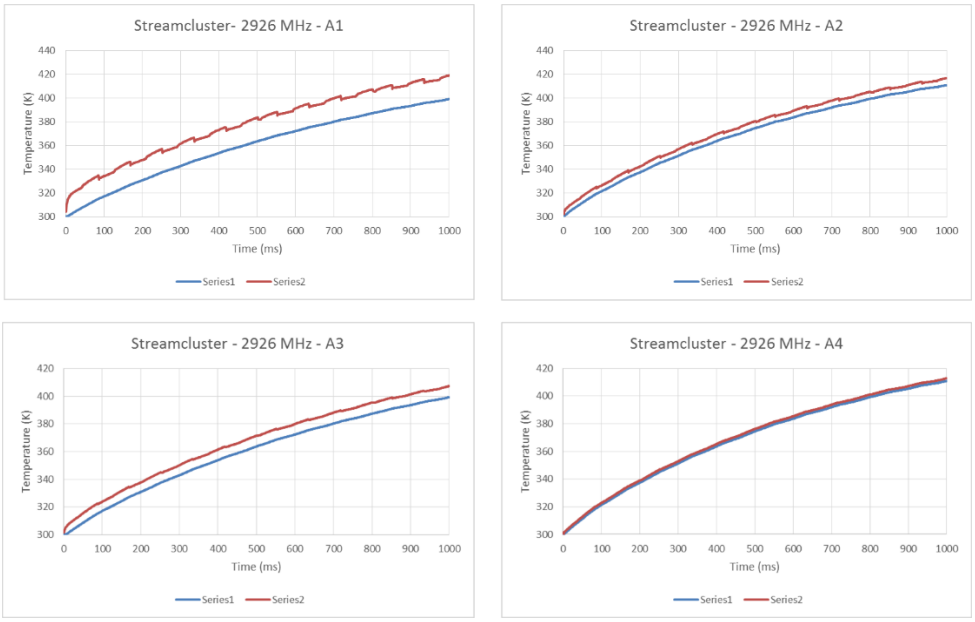


Figure 4.22: Thermal trace for $T_{junction}$ and T_{case} for each configuration at 2926 MHz

Temperature violations are reported at 380, 403, 486 and 444ms respectively. We can see that from this point of view, this simulation set results in smaller errors, despite the rapid ascension of temperature in the chip. This fact steers us toward the theory that abrupt temperature spikes are not that influencing in the models if they are not coupled with high power densities.

4.3.4 Overestimation Method

A common practice in thermal modelling, is to assume worst case, per-core, power consumption, evenly distributed in the core die without involving any power modelling tools. This method is based on the idea that conclusions made with a worst case scenario basis, can only deviate to the better in practical circumstances. In order to test that theory, we used the previous A4 configuration for each benchmark, only in this case we calibrated the power values from McPAT so that the curve from A4 would approximate the thermal trace of A1. In order to quantify how the calibrated curve is adjusted, we aimed to minimize the average error value between the two thermal traces for the window of 1000ms. This was achieved through a simple trial and error process, assigning a scaling factor to the power values, computing the corresponding temperature trace and consequently the average error between this new trace and that of A1. After a small number of attempts the optimal scaling factor was determined.

It is not readily seen, but this approach yields better overall results than attempting to adjust the A4 curve to be steadily over that of A1 (thus resulting to a worst case scenario). First of all, no amount of reasonable calibration can ensure that the thermal trace from A4 is above the trace of A1 completely. In addition, attempting to ensure that the calibrated trace will supersede the other even for three quarters of the transient time window we examine, leads to very large temperatures both in the end of the transient and in the resulting steady state region. These facts, only seem to

reinforce our belief, that the thermal model represented by A4 is unable to model transient effects accurately. To be more thorough, let us examine the thermal traces for the best fitting curves for each benchmark presented in Figure 4.23.

It is readily observed that any kind of abrupt spike in temperature is still elusive for the model we examine. High power densities that are present strongly in the serial phases of simulations are only "shadowed" at a distance. Less fluctuating temperature rises seem to be simulated adequately with the adjustment we made. However, benchmarks like Bodytrack, characterized by oscillating power values create thermal traces with analogous temperature swings that are simulated only in an average value approach.

It is also worth noting, that due to big differences in thermal resistance and capacities of the overall models (RC factors), curve fitting is imprecise even for values resulting from stable power consumption, due to the variability of rise time and resulting steady state temperature. Nevertheless, for such cases, adjustments that ensure that the worst case scenario model will err only on the safe side, can be effected. This is only emphasized because it does not hold true for fluctuating power values.

Of course, all the previous inaccuracies (thermal profiles, vertical heat spreading etc.) of the A4 model are also in effect here regardless. This derives from the fact that these are indigenous to the thermal configuration when contrasted with A1.

A reasonable argument could be made as to why we opted to compare models 1 and 4 neglecting all the others. The selection of A1 is pretty much self-evident, it is the model we consider to accurately model the real phenomena involved. The reason A4 was selected on its own, is because it is the most used thermal model in the literature. To the best of our knowledge, thermal interface materials are generally ignored and a chip design employing the use of a heat spreader without interface materials is not existent in the literature. Hence, we chose to compare a frequently used model for configurations like the one we presented, to our own.

From another perspective, even if we assume a very efficient design that results in completely homogeneous power distribution across the core die, inaccuracies will still be incurred as a consequence of improper heat spread modelling. This case is actually the comparison between A3 and A4 which we know that is still burdened with certain weaknesses even in the ideal case of total uniform power allocation on core. These weaknesses have been highlighted in a small, four core chip with small capacity at core utilization diversity. In a chip with many cores and many diverse operation schemes available, the inaccuracies between the two models will only multiply. This derives from the fact that any concentrated heat source results in erroneous results should the interface materials be neglected, regardless of the origin or area of the hot spot. Thus, even if different elements of the core are more active, depending on application and workload conditions, or different cores in the same chip are more active, the conclusions we have reached throughout this chapter still hold true.

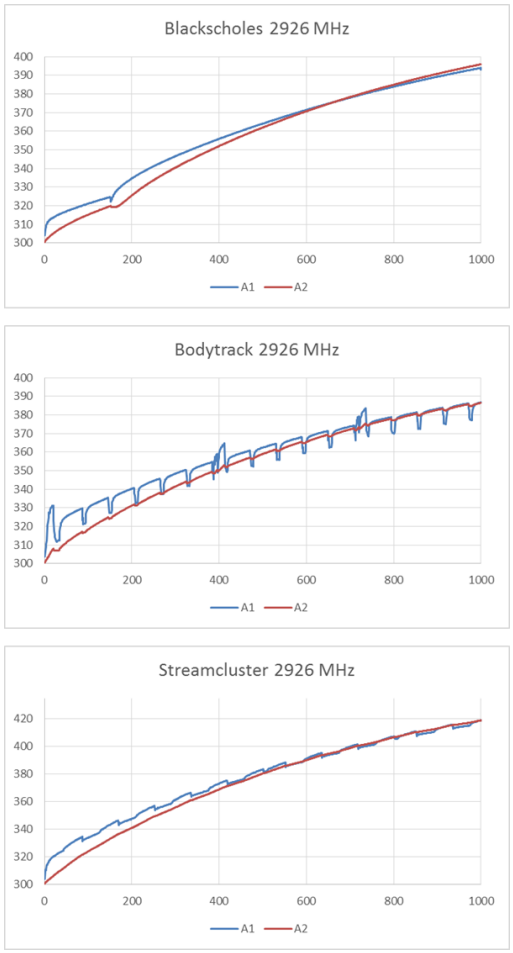


Figure 4.23: Temperature traces between A1 and calibrated A4 configurations for all benchmarks

4.4 Summary – Conclusion

In this chapter, a frequently used thermal model in the literature has been tested against our own. The factors differentiating the results deriving from the compared models are the modelling, or not, of interface materials, and the modelling, or not, of uneven power allocation in the core die. To showcase the importance of each of these factors, additional configurations employing one but not the other were examined. In order to explore a variety of scenarios, different benchmarks with different core and component utilization, at different frequencies, were used. Furthermore, a more accurate, calibrated version of the thermal model we theorized would be imprecise, was also tested.

In order to provide a complete analysis of all the factors weighing on the results, each test case was approached from many different perspectives. As expected, we found that thermal models neglecting heat distribution among cores and thermal interface materials have difficulty modelling transient phenomena tied to high power densities. In addition, the resulting steady state values generally lead to underestimations.

One of the most important facts in the whole analysis presented in this section, was the impact of operating frequency on the error values reported. We observed that with increasing frequency values, the errors in thermal modelling grew rapidly. This seems to indicate that in older, lower frequency chips characterized by inferior power densities, the inaccuracies between the thermal models would be almost imperceptible. Another mitigating factor would be the single core nature of older chips or uniform operating scenarios frequently employed. This derives from our conclusion that uneven power allocation resulting from any factor creates transient phenomena not captured in the tested thermal model.

Moreover, we argued the validity of the weighting factors that distinguish our model as more accurate. Regarding TIMs, to the best of our knowledge, no chip employing the use of heat spreader can form a spreader – die joint without interface materials. Regarding heat spreading, assuming that other areas, meaning other components of the chip are more active, is irrelevant to our conclusions. The same phenomena would be present. Assuming a greatly efficient design, resulting to homogeneous power allocation in each core die, certainly results to smaller errors, but still effects inaccuracies due to uneven core utilization on a chip-level basis. Even if that variable becomes uniform, still the tested model would not be perfect as it would lead to higher temperature gradients and smaller steady state values. Furthermore, in any case, vertical heat spreading is highly erroneous when neglecting TIMs.

To make matters worse, we argue that all these inaccuracies will be further exacerbated in sprinting platforms. Sprinting results in higher frequencies and/or irregular core utilization, depending on workload conditions, the very factors intensifying power densities and hot spot creation. In such cases, the effect of TIMs interacting with uneven heat distribution is multiplied.

On a final note, PCMs usually employed in sprinting systems are influenced by all of the factors we mentioned. Because the variables involved are closely related and interacting, it is difficult to clearly distinguish cause and effect. Even through the analysis and examples presented, a degree of uncertainty regarding the results of ignoring a factor might still be justified. Nevertheless, it is proven beyond doubt that ignoring TIMs and heat distribution among cores can lead to very big inaccuracies in chips intending to use frequency and/or parallel sprinting and PCMs, which actually are the object of this thesis. Consequently, our thermal model not only models the use of PCMs in the layer stack, it also does so with accuracy not captured by other thermal models, for the chip configuration we examine.

CHAPTER 5

5.1 Phase Change Material Exploration Objective

The use of phase change materials in systems utilizing computational sprinting has already been analyzed in previous chapters. It is a relatively recent innovation and although there is already research with interesting results publicized, a number of factors have yet to be determined. One could say that most of the research already conducted, basically exhibits that the use of said materials can be beneficiary under specific circumstances. For example, in [9], the results of using a PCM that melts at 80 °C interposed between the silicon and heat spreader is shown. A small exploration follows, exhibiting the effects of thermal conductance and thickness of the phase change material. To the best of our knowledge this work is probably the most thorough regarding the specific use of PCMs in computational sprinting. In most of the other cases, a specific material with fixed properties is used and the corresponding benefits are presented.

In this thesis, instead of using a specific material to boost our system, we opted to explore a variety of configurations using PCMs. The idea is to attempt to determine the optimal characteristics for such a material. The physical properties we chose to explore was the melting point, thickness and placement. The latter is generally chosen arbitrarily in the literature but we believe that determining the best position for the PCM layer in the chip stack, is a goal worth pursuing. We chose to assume that we can place a layer of phase change material, however thick, anywhere in the chip stack. Of course, although we can simulate such theoretical configurations using our framework, they might not be achievable in real devices. Still, we deemed that the results of such an exploration might provide valuable insight or even a goal worth pursuing by the industry of packaging.

5.2 Simulation Methodology

In order to test various PCM configurations we generally retained all the configuration parameters described in Chapter 4 including, of course, the thermal interface materials. The difference in this chapter was that we created four different chip stacks involving the use of phase change material at different locations in the vertical direction. These layouts named P1 to P4 are presented in Figure 5.1. Note that no height (or thickness) is listed for the PCM layer since height is one of our control variables. We also used the same recorded benchmarks as presented in Chapter 4, that is, Blackscholes, Bodytrack and Streamcluster at 2926 MHz

In the first part of simulations conducted, the configuration listed as P1 with a material that melts at 60 °C and is 100μm thick, was simulated for the Blackscholes benchmark, without thermal interface materials at first, with uniform heat distribution afterwards and then both without TIMs and with uniform heat distribution. These simulations are presented to showcase the impact of the conclusions from Chapter 4 in PCM enabled systems.

Afterwards, we simulated each benchmark for each configuration P1 through P4 until the first temperature violation occurs. Since operating temperatures for a chip generally range from 40°C to 100°C, and research shows that materials with corresponding melting temperatures and

adequate thermal conductivity exist, we tested all the range from 40 to 90 degrees with 5 degree increments. Testing higher than 90 degrees Celsius was considered trivial since we know that PCMs display a latency effect in their use and we further expect that even the range of 80 to 90 degrees will not show any particular gain.

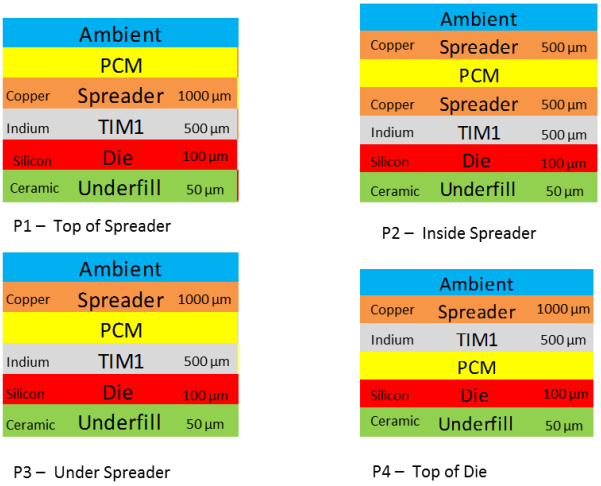


Figure 5.1: Configurations used to determine optimal placement of the PCM layer

Regarding thermal conductivity, conductivity enhancement techniques are mentioned in [9], with which, copper-PCM compounds can achieve values up to 106 W/mK. For our simulations, we selected a more conservative value of 75.4 W/mK assuming a pretty high PCM fraction in the composite. Regarding the heat capacity and the apparent heat capacity method, we need to define two values: the volumetric heat capacity corresponding to the solid and liquid phase, and the volumetric heat capacity corresponding to the transient phase, that is, the phase change region. The first of the two, we consider to be unaffected form the enhancement technique using copper since the volumetric heat capacity for copper, a value of $1{,}57 * 10^6$ J / m^3 K, is approximately identical to that listed in [9] for the PCM, $1{,}56 * 10^6$ J / m^3 K. For the phase change region, we also used a smaller value than the one proposed, $244 * 10^6$ J / m^3 K instead of $305 * 10^6$ J / m^3 K, to reflect an analogous reduction to the fraction of copper in the composite. Since phase change materials generally do not have big differences in volumetric heat capacity values, the same pair was used for all melting temperatures even though different melting temperatures correspond to different materials.

In essence, we used a theoretical material with fixed thermal conductivity and volumetric heat capacities but variable melting temperature. The reason for this choice is the fact that we do not want to test specific materials rather than test the impact of different melting points in our various configurations while keeping the other values constant. The sole reason for the previous analysis was to establish that the values we used are reasonable and achievable for all materials within the examined melting temperature region.

This exploration, was conducted for PCM thicknesses of 100 and 200 µm. Afterwards, having observed a trend in the simulation results, we chose the most promising configuration

regarding placement and melting temperature and continued testing for phase change material thicknesses up to 700 μm with 100μm increments. Further exploration, involving thicker layers was avoided since 700 μm of PCM material results in an approximate increase of 100% of the chip thermal resistance to the ambient. This observation is more thoroughly analyzed in subsection 5.3.3.

5.3 Results and Discussion

5.3.1 Thermal Interface Materials and Heat Distribution Impact

As an extension to the results demonstrated in Chapter 4, a set of simulations was conducted in this subsection to showcase the inaccuracies introduced by omitting thermal interface materials and disregarding uneven heat distribution in the chip components. To this end, we simulated the configuration listed as P1 for the Blackscholes benchmark with a phase change material that melts at 60 degrees Celsius and rests on top of the heat spreader. The thickness of the material was 100μm. This configuration was simulated with three variations. In the first, the thermal interface material was omitted. Next, the thermal interface material was included in the chip stack, but the heat distribution in the chip was considered homogeneous. Lastly, both inaccuracy factors were combined.

Regarding simulation times, let us note that the baseline configuration, where both the thermal interface materials and fine-grained heat distributions were considered, the first temperature violation occurred at 718 ms. In the case where no thermal interface materials were considered but the heat distribution was still fine-grained, the first temperature violation occurred at 781 ms. Maintaining TIMs and considering homogeneous heat distribution resulted in a staggering 908 ms. Combining the absence of TIMs with homogeneous heat distribution resulted in 847 ms.

These overestimations of simulation time are consistent with the analysis presented in Chapter 4. Disregarding thermal interface materials and proper heat distribution leads to completely different temperature profiles. The crucial factor not addressed is the formulation of hotspots due to more active components. Failure to address this factor will result in temperature underestimation and thus overestimation of simulation time.

In addition, as we have stated earlier, not modelling hotspots and heat flow in the vertical direction correctly, leads to completely different thermal profiles that introduce a collection of inaccuracies. To prove this point, let us look at Figure 5.6 where the thermal snapshots at the time of the first temperature violation are presented. These snapshots depict the PCM and chip layer for each of the four configurations.

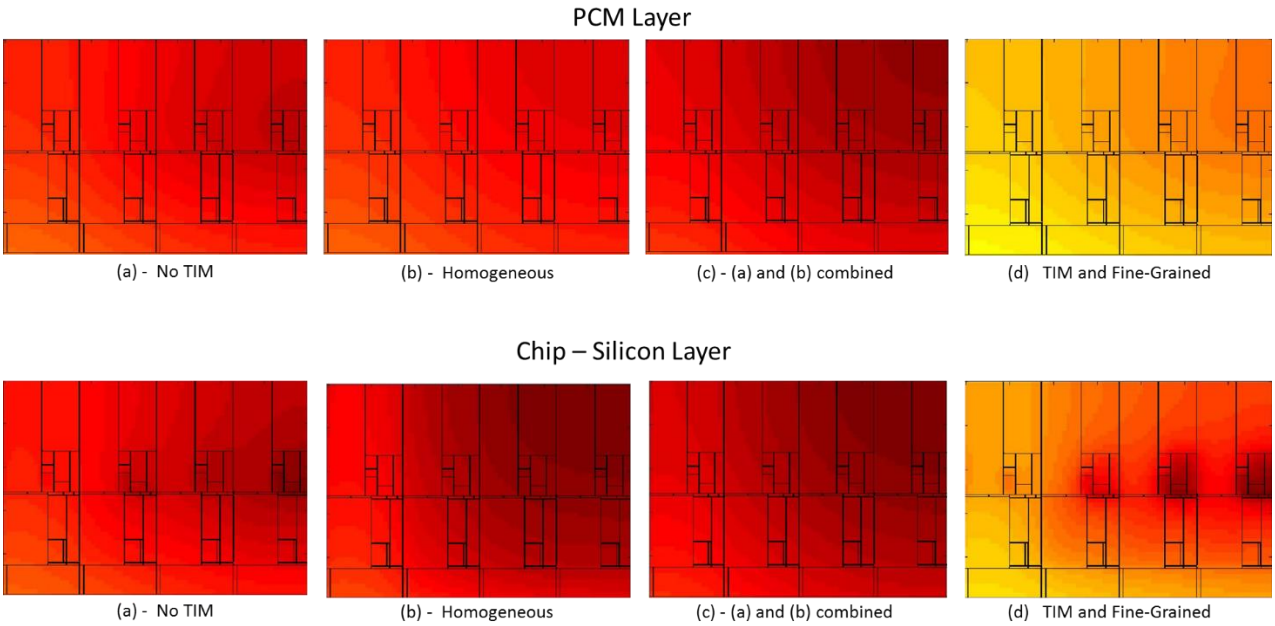


Figure 5.2: Thermal Snapshots for each configuration at the time of first temperature violation

While not many differences can be observed between the inaccurate configurations, the thermal snapshot from our proposed model is completely different. One has only to look at this figure to understand the inaccuracies in terms of PCM internal energy and percentage melted that will result, let alone the simulation time which we already addressed.

As expected, the inaccuracies outlined in Chapter 4 are exacerbated when PCM materials are involved. More to the point, this fact was determined using a mere 100μm of phase change material and a mere four core platform. The implications for more complex designs involving more PCM and numerous cores are quite dire. However, we felt that further demonstrating such inaccuracies by using different configurations and benchmarks would be redundant.

5.3.2 Melting Point and Placement Exploration

As explained earlier, in order to explore the behavior of systems with phase change materials of different melting temperatures and placement in the chip stack, configurations P1 to P4 were tested with melting points ranging from 40 °C to 90 °C for Blackscholes, Bodytrack and Streamcluster, as presented in Chapter 4. Also, the simulations were conducted for PCM thicknesses of 100 and 200 μm. The results are evaluated with respect to the time of the first reported temperature violation. To elaborate, each benchmark was simulated as a baseline without any PCM layer until the first temperature violation is reported. Afterwards, each PCM configuration was similarly simulated and the difference between each configuration and the baseline was logged. The extra time, in milliseconds, that was gained due to the presence of the phase change material for each setup, is presented in detail in Table 5.2. Note that a negative value

designates that the PCM configuration actually performs worse than the baseline. For clarity, the best performing melting point for each configuration is highlighted with green color and the best performing configuration for each benchmark is highlighted with red. As best performing configuration, we designate the one in which the maximum extra simulation time was reported.

Blackscholes	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	135	131	133	136	138	131	102	79	17	17	17
		P2	137	132	131	131	133	125	97	81	12	12	12
		P3	111	114	114	116	117	105	78	61	2	-1	-1
		P4	105	108	113	114	113	98	74	63	44	27	31
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	243	251	259	269	272	251	201	122	34	34	34
		P2	238	246	251	259	261	241	187	142	24	24	24
		P3	208	214	223	229	227	185	146	102	3	-2	-2
		P4	203	211	222	223	214	171	139	109	81	53	31
Bodytrack	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	92	92	92	92	99	92	90	52	19	19	19
		P2	98	97	92	92	92	92	90	69	13	13	13
		P3	92	92	92	98	92	92	92	69	1	1	1
		P4	92	92	98	92	92	92	92	97	59	28	35
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	286	307	313	326	341	106	92	66	49	49	49
		P2	309	309	312	318	331	106	92	88	43	43	43
		P3	109	110	272	110	108	102	92	89	2	2	2
		P4	108	110	284	112	108	103	100	97	75	64	56
Streamcluster	100	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	88	91	91	92	91	86	82	9	7	7	7
		P2	90	91	91	90	88	83	81	11	5	5	5
		P3	83	83	83	83	83	81	79	21	1	1	1
		P4	83	83	83	83	82	81	81	35	32	21	32
	Thickness												
	200	cfg	40	45	50	55	60	65	70	75	80	85	90
		P1	163	170	171	171	163	155	144	16	15	15	15
		P2	162	162	166	166	157	148	141	16	10	10	10
		P3	145	147	151	149	141	119	101	22	1	1	1
		P4	145	146	148	144	137	115	104	87	81	62	35

Table 5.1: Time (ms) gained until first temperature violation for all simulations conducted

Regarding overall trends and general performance, we can see that configuration P1 is in all cases the best performing configuration. In addition, the melting points in the region of 50 to 60 degrees Celsius are dominant in all simulations regardless of configuration or benchmark. However, the latter (60) has most commonly the best performance and in cases where it is not the best performing melting point, the difference is quite small. Notably, in all cases except Streamcluster, the melting point in which the best configuration, P1, showed the best benefit, was that of 60 degrees Celsius. In Streamcluster, the best performance was achieved for 55 degrees but here too the 60 degrees case, is only slightly worse.

In order to get a better grasp on what diversifies the results in all these configurations, we created Tables 5.2 and 5.3. In Table 5.2 we demonstrate for each setup the percentage of the PCM

that has melted at the time the first temperature violation was reported. Table 5.3 shows a different metric at the same time instant, the total energy absorbed by the PCM layers during the simulation. For reference, in both figures, the best performing configurations, in terms of time gained, were also highlighted.

			40	45	50	55	60	65	70	75	80	85	90
Blackscholes	100	P1	100.00	99.92	99.83	100.00	99.91	95.62	65.51	26.03	0.00	0.00	0.00
		P2	100.00	99.97	99.96	99.98	99.66	96.04	67.10	31.87	0.00	0.00	0.00
		P3	100.00	99.99	100.00	99.90	99.97	92.82	61.40	28.45	0.11	0.00	0.00
		P4	99.63	100.00	100.00	100.00	99.23	90.02	62.49	39.69	17.10	4.86	1.26
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	99.36	99.96	99.99	99.95	99.90	90.15	62.36	20.39	0.00	0.00	0.00
		P2	99.00	100.00	99.97	99.90	99.98	90.47	61.86	30.03	0.00	0.00	0.00
		P3	99.99	99.00	99.94	99.94	98.25	80.69	53.57	23.46	0.07	0.00	0.00
		P4	99.90	100.00	99.40	99.92	96.21	77.05	54.55	32.87	14.65	4.15	0.63
			40	45	50	55	60	65	70	75	80	85	90
Bodytrack	100	P1	99.99	100.00	99.95	99.00	99.93	85.58	58.35	8.94	0.00	0.00	0.00
		P2	99.83	100.00	99.99	99.97	99.96	87.74	61.81	18.58	0.00	0.00	0.00
		P3	100.00	100.00	100.00	100.00	99.97	93.96	66.43	26.29	0.00	0.00	0.00
		P4	99.90	100.00	99.85	99.84	99.35	87.53	67.84	44.65	12.29	0.10	0.17
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	100.00	99.99	99.98	99.96	99.94	58.29	29.39	4.06	0.00	0.00	0.00
		P2	100.00	100.00	99.66	99.80	99.98	61.37	33.31	10.54	0.00	0.00	0.00
		P3	99.93	99.96	99.98	95.33	77.06	61.67	33.36	14.35	0.00	0.00	0.00
		P4	100.00	100.00	100.00	99.92	75.12	57.25	39.91	25.21	10.54	0.00	0.06
			40	45	50	55	60	65	70	75	80	85	90
Streamcluster	100	P1	100.00	99.86	99.98	99.94	99.04	89.57	70.61	2.80	0.00	0.00	0.00
		P2	99.00	99.99	99.98	100.00	99.82	90.65	72.51	6.92	0.00	0.00	0.00
		P3	99.00	99.99	99.98	100.00	99.43	89.72	72.21	18.20	0.00	0.00	0.00
		P4	99.85	99.88	99.94	99.93	99.75	87.07	71.39	41.73	18.89	5.74	1.31
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	99.87	99.29	99.98	99.80	94.87	81.72	66.44	0.85	0.00	0.00	0.00
		P2	100.00	99.94	100.00	99.98	95.44	81.73	68.09	2.68	0.00	0.00	0.00
		P3	99.97	99.42	99.98	99.24	91.49	75.11	55.64	8.02	0.00	0.00	0.00
		P4	99.97	99.99	99.87	98.55	87.95	73.68	57.64	39.06	21.25	7.40	0.51

Table 5.2: Percentage of melted PCM at the time of temperature violation

The percentages shown in Table 5.2, seem to follow anticipated trends. High melting points exhibit small percentages of PCM melted or even zero, which explains their low performance. Accordingly, low melting points show very high fractions of the total of the PCM having melted. Note, that this metric describes only if all the PCM cells have melted. For example, a case where all the cells but one, have just melted and a case where all the cells but one, have not just melted, but have also skyrocketed to 100 degrees Celsius, would show the same percentage.

Consequently, this metric is only an indication of how much of the available heat capacity in the phase change region has been leveraged. In addition, as we can see from the Figure, the configuration performance does not seem to have an absolute relationship with this metric. While the best performing configurations are, without exception, characterized by very high percentages

of melted phase change material, these same configurations do not exhibit the highest values in the figure.

In order to get a better sense, of how much heat has actually been absorbed by the PCM layer, we created Table 5.3. In this case, we can see the energy stored in all the PCM cells since the beginning of the simulation to the time of the first temperature violation. This metric accounts for all thermal energy regardless of whether phase change occurs or not and at which percentage. It is worth noting that this metric differs from the previous in the sense that it also accounts for the behavior of the material before and after the phase change region. This is quite evident noticing that configurations in which all of the phase change material has melted, result in different total energy values.

			40	45	50	55	60	65	70	75	80	85	90
Blackscholes	100	P1	2.661	2.713	2.661	2.666	2.663	2.558	1.864	0.967	0.358	0.358	0.358
		P2	2.660	2.661	2.663	2.665	2.658	2.569	1.902	1.104	0.360	0.360	0.360
		P3	2.667	2.662	2.664	2.662	2.663	2.491	1.770	1.023	0.362	0.358	0.358
		P4	2.676	2.687	2.687	2.688	2.666	2.445	1.813	1.296	0.782	0.502	0.425
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	5.309	5.328	5.331	5.328	5.315	4.850	3.578	1.673	0.715	0.715	0.715
		P2	5.327	5.323	5.324	5.322	5.317	4.866	3.556	2.124	0.714	0.714	0.714
		P3	5.310	5.312	5.312	5.308	5.218	4.395	3.168	1.814	0.708	0.700	0.700
		P4	5.350	5.362	5.348	5.347	5.156	4.260	3.243	2.268	1.453	0.977	0.806
			40	45	50	55	60	65	70	75	80	85	90
Bodytrack	100	P1	2.634	2.637	2.636	2.636	2.634	2.303	1.687	0.567	0.358	0.358	0.358
		P2	2.628	2.637	2.637	2.637	2.635	2.356	1.770	0.793	0.359	0.359	0.359
		P3	2.650	2.640	2.642	2.632	2.640	2.508	1.883	0.972	0.377	0.222	0.362
		P4	2.663	2.661	2.635	2.659	2.644	2.372	1.925	1.410	0.671	0.227	0.386
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	5.325	5.332	5.329	5.327	5.324	3.294	2.018	0.904	0.719	0.719	0.361
		P2	5.320	5.323	5.309	5.317	5.326	3.442	2.201	1.211	0.723	0.723	0.723
		P3	5.202	5.198	5.317	4.966	4.133	3.463	2.205	1.387	0.704	0.704	0.704
		P4	5.252	5.246	5.364	5.286	4.080	3.284	2.528	1.899	1.264	0.460	0.717
			40	45	50	55	60	65	70	75	80	85	90
Streamcluster	100	P1	2.663	2.654	2.658	2.657	2.635	2.471	2.030	0.414	0.353	0.353	0.353
		P2	2.660	2.659	2.660	2.661	2.654	2.498	2.076	0.513	0.355	0.355	0.355
		P3	2.659	2.658	2.662	2.661	2.648	2.465	2.067	0.795	0.358	0.358	0.358
		P4	2.688	2.690	2.691	2.691	2.651	2.411	2.032	1.352	0.834	0.534	0.439
	Thickness												
			40	45	50	55	60	65	70	75	80	85	90
	200	P1	5.296	5.286	5.303	5.285	5.046	4.451	3.759	0.733	0.698	0.698	0.698
		P2	5.293	5.294	5.302	5.297	5.074	4.455	3.843	0.822	0.699	0.699	0.699
		P3	5.292	5.289	5.294	5.249	4.891	4.139	3.255	1.075	0.701	0.701	0.701
		P4	5.356	5.355	5.345	5.266	4.780	4.123	3.389	2.557	1.770	1.159	0.812

Table 5.3: Thermal energy (J) stored in the PCM at the time of temperature violation

Similarly, in this case, we can also see that even though the best performing configurations are characterized by large sums of absorbed energy, these configurations do not exhibit the highest values observed. On the other hand, one can also notice that melting temperatures below 60 degrees Celsius are characterized by small differences in all the metrics presented. Simulation times are only slightly lower, percentages vary no more 0.2 % and thermal energies no more than 5%. In contrast, higher melting points have differences consisting of lower percentages of material melted, lower thermal energies, and, as expected, quite lower simulation times.

The previous analysis seems to indicate that there is a direct relationship between the percentage of the melted material, the total energy absorbed and the performance of the current configuration even though this relationship is not absolute. For the most part, these results are within our expectations with one small difference. The fact that configurations with higher percentages of phase change material melted and total energy absorbed, might actually perform worse than others. This fact illustrates that leveraging the extra thermal capacity that phase change materials offer is not enough, in order to achieve maximum performance. Simulation results show that this added thermal capacity is more useful when exploited at the right time (in essence meaning at critical temperature and heat generation combinations).

With respect to the different configurations P1 through P4 used in the simulations, it was earlier noted that P1 had the best performance in all benchmarks. This we expected because all cases except P1 generally add thermal resistance between the chip active components and the copper spreader. To elaborate, the spreader, having very high thermal conductivity, primarily buffers abrupt temperature increases and spreads the accumulated heat at a rapid pace in the vertical and lateral direction. The only downside, that prohibits the spreader from doing this for longer periods of time, is the low thermal capacity of copper. This means that the spreader not only propagates heat quickly but also increases in temperature rapidly, thus reducing heat flow from the active components (heat flow is linearly dependent on the temperature difference between interfaces).

On the other side of the spreader, since our system does not employ a heat sink, heat flow to the ambient is generally slow despite the high conductivity of copper. This is a direct result of the low thermal conductivity of air. As a result, we have the following: the copper spreader can effectively spread and abduct heat but is hampered by its low thermal capacity, on the other side, air can effectively absorb all heat generated, due to its constant replacement and abundance in the ambient, but is hampered by its low thermal conductivity. A promising idea would be to interpose an intermediate component that balances between the two extremes. In our case this intermediate component is the PCM layer. It has higher thermal conductivity than air, but can store a respectable amount of heat in the phase change region before its temperature starts rising again. This is configuration P1.

In cases P3 and P4, what happens essentially, is that we add a better heat storage than copper close to the active components. This, as we have seen, increases the system's buffering abilities greatly for the duration of the phase change region, but before and after, the resulting thermal model is worse off than we started because of the increased thermal resistance to the spreader. In addition, because the thermal conductivity of the PCM is not very high, heat is generally focused in the active areas, and after melting the PCM material in those areas, the chip now has one more layer that cannot efficiently absorb heat from the more active regions. In contrast, without interposing the PCM below the spreader layer, only the thermal interface material layer is interposed between the spreader the active components.

Case P2 is actually a hybrid between the P1 and P3, P4. It was simulated as a special configuration, dividing the spreader to two equal layers and interposing the PCM between them,

in an effort to surround the phase change material with highly conductive material. However, the overall performance was not better than that of P1.

All in all, configuration P1 undoubtedly exhibited the best performance in all benchmarks. This is encouraging since this configuration is the only one that has already been shown to be easily achievable. For this configuration, a combination of a melting point of 60 °C achieved the best result. Although for Streamcluster that was not the case, the difference was actually quite small. We intend to use this melting point as the best for the next group of simulations not only because of the performance gains already discussed, but also because using low melting points involves the risk of the material melting in non-critical conditions as a result of other computational activities and non sprinting methodologies. An added benefit to this fact is that configuration P1 places the phase change material as far from the active components as possible, resulting in lower idle temperatures and more sprinting headroom available. In conclusion, we feel that the combination of configuration P1 along with a melting point of 60 °C represents the best choice not only from the simulation data we have already seen, but also with regard to actual system considerations.

5.3.3 Thickness Exploration

In this subsection, we conducted a set of simulations, for each of the benchmarks already discussed, with PCM layers at the top of the copper spreader that melt at 60 °C. The variable addressed here more thoroughly, was that of the PCM layer thickness. In order to explore the benefits of adding more and more phase change material to the system, in each simulation, we added another layer 100 μm thick, up until 700 μm. Again, the simulations were conducted until the first temperature violation occurred. The results, in terms of the total simulation time, are presented in Table 5.4.

PCM Thickness	0	100	200	300	400	500	600	700	μm
Blackscholes	580	718	852	984	1115	1242	1368	1491	s
Bodytrack	626	725	967	1000	1201	1208	1519	1529	s
Streamcluster	380	471	543	629	713	796	901	985	s
Thermal Resistance increase relative to baseline									
	0	15.65	31.30	46.95	62.61	78.26	93.91	109.56	%
Simulation Time increase relative to baseline									
Blackscholes	0	23.79	46.90	69.66	92.24	114.14	135.86	157.07	%
Bodytrack	0	15.81	54.47	59.74	91.85	92.97	142.65	144.25	%
Streamcluster	0	23.95	42.89	65.53	87.63	109.47	137.11	159.21	%

Table 5.4: Total simulation time before first temperature violation

Additionally, two other metrics are presented in Table 5.4. The first of the two is the increase in thermal resistance, between core components and ambient, relative to the baseline configuration we discussed earlier, as a result of the added phase change material. The second

metric represents the simulation time gained relative to the baseline. Naturally, this metric does not provide any further information than the absolute simulation times. However, it was presented in this format, in order to provide a straightforward comparison between the percentage increase of the thermal resistance and the percentage increase of the computation time. It is of course worth noting, that the percentage increase in time, before the first temperature violation, is dependent upon the benchmark, the number of active threads, the type of computation, and the operating frequency of the cores, whereas, the increase in thermal resistance is fixed for a specific material and layer thickness.

That being said, even though we observe that for a mix of quite different benchmarks, the time gain is bigger than the corresponding thermal resistance increase, other factors must also be considered.

The most important is the impact of the added thermal resistance to the chip. In our simulations, we assumed that before the benchmark was issued, the system was idling. For idling systems, core temperatures are only slightly higher than the ambient. For this reason, each simulation was initiated with all components at 300 K which is slightly above room temperature. However, when the resistance between core and ambient increases, so does the corresponding difference in temperatures. More to the point, this difference in temperature, is also dependent on the heat flow and thus power consumption of the active components. To be exact, there is a linear dependence of the core temperature (relative to the ambient) and both the thermal resistance and the power load. In essence, the effect of increasing the thermal resistance is exacerbated under heavy load conditions.

For example, assume an arbitrary, steady state power consumption that leads to the core temperature averaging at about 40 °C. This indicates a temperature difference of 15 degrees between core and ambient. When using a 700μm thick PCM layer, the thermal resistance between core and ambient approximately doubles, that means that the new difference would be 30 degrees, and the new core temperature 55 °C. Keeping in mind that our phase change material starts melting at 60 degrees Celsius, even though it rests on top of the heat spreader, where temperature levels would be somewhat lower, these results indicate that, depending on previous conditions, performance benefits might be a lot smaller and maybe not worth the extra material.

In comparison, in idle conditions, where the core is only 2 – 3 degrees hotter than the ambient, doubling the thermal resistance will result in the core being 4 – 6 degrees hotter than the ambient, which is barely noticeable.

From another point of view, adding PCM material greatly increases the time constant of the system. This happens because the time constant is dependent on both the thermal resistance and the thermal capacitance. Increasing the time constant means that the system takes more time to rise in temperature but simultaneously takes more time to cool down.

Combining all the previous points, we can see that adding PCM material alters the thermal profile of the chip. While actually stating the obvious, the implication is that the thickness of the phase change material has to be chosen depending on the type of tasks and activity we intend to perform in a usual basis. For example, chips that usually perform short computational tasks that

are issued frequently, would opt to use little PCM. On the other hand, chips that are tasked with more elongated computational tasks that are issued on an infrequent basis, would be most benefited by using large quantities of phase change material despite the increase in thermal resistance.

5.4 Summary – Conclusion

In this chapter, we further proved the conclusion we derived from the previous one, that thermal models that do not account for thermal interface materials and heterogeneous heat distribution, are not suitable for simulating sprinting enabled systems augmented with PCM material.

Afterwards, we performed a series of simulations and determined that placing phase change materials on the top of the heat spreader not only guarantees the best performance, but is also an easily achievable configuration and less susceptible to be impacted by the previous state of the system.

In collaboration with this configuration, we found that a melting point of 60 °C for the phase change material is the best available choice. This choice demonstrated the best performance at the majority of the cases examined, and lags only slightly in all other cases. Simultaneously, this melting point is high enough to preclude the melting of the PCM in non-critical conditions, especially when placed on the top of the heat spreader which is the cooler part of the chip stack.

Furthermore, we saw that increasing the thickness of the PCM layer resulted in analogous benefits for all the sizes we examined. However, we argued that this variable represents a trade-off. This trade-off depends on the type of computational tasks the system is likely to perform more often. As a general guideline, long-lasting tasks followed by long idle periods, require big quantities of phase change material in order to be able to sprint for the whole duration of task. Short tasks followed by short idle periods, require small quantities in order to be able withstand the generated heat but also be able to cool down in the short time available.

CHAPTER 6

6.1 Conclusion

In this work, we developed a simulation framework for systems that utilize computational sprinting and are augmented by phase change materials. The framework involves performance simulation by virtue of the Sniper Simulator, power consumption simulation via McPAT and thermal simulation using 3D-ICE. These three distinct simulation tools have been integrated to a unified framework written in Python.

The framework implements a Python server-client model with the server satisfying simulation requests indefinitely. Simulation requests can be issued in the form of a queue. During each simulation, the framework handles the logging of statistical data and simulation information into suitable files. These files are properly archived at the end of each simulation. Further processing of outputted data is achieved through independent Python scripts and visualization of thermal and power maps through the use of Matlab.

Integration of the previously mentioned simulation tools into the Python framework involved a number of additions and modifications in the source code of each tool. However, in the case of 3D-ICE, more extensive development took place in order to add to the simulator the ability to properly model phase change materials. This was achieved using the *apparent heat capacity* method, that is, assigning a non-linear volumetric heat capacity to material cells if a set of specific circumstances are met. This method was further extended in order for the simulator to be able to model layer of PCM with different melting points both in the vertical direction and across the cells of the same layer.

Afterwards, we demonstrated that commonly used thermal models are unsuitable for applications involving computational sprinting and phase change materials due to highly irregular heat distribution. In addition, we proposed a set of modifications, involving thermal interface materials and modelling fine-grained power consumption, in order to provide a model better suitable for the systems we intend to research.

Lastly, we performed an exploration regarding optimal melting point and placement in the vertical stack of phase change materials. More specifically, we performed tests using three benchmarks from the parsec suite, Blackscholes, Bodytrack and Streamcluster, eleven melting points ranging from 40 to 90 degrees Celsius with 5-degree increments and four different configurations each placing the PCM layer at a different location of the chip stack.

Results showed that placing the PCM on top of the heat spreader and using a melting point of 60 degrees Celsius provided the best results for all benchmarks. Consequently, we used this optimal configuration and started adding more PCM material in 100μm increments, in order to test the effect of the added PCM to the system. As expected, simulations showed that increasing the thickness of the PCM results in an analogous increase in computation time before the first temperature violation. This observation held true for thicknesses up to 700μm and indicates that by varying the material thickness we can effectively alter the system's time constant thus altering maximum sprinting time and necessary rest period.

6.2 Future Work

In this diploma thesis, only a small fraction of the capabilities of the simulation framework that was constructed were used. Through the use of the Sniper Simulator, a powerful tool that is highly customizable, McPAT and 3D-ICE, a great range of configurations and architectures can be simulated. However, despite porting a number of variables from each tool to a single control script, routine altering of these simulation characteristics can be quite tedious. In addition, using the framework necessitates a basic understanding of Python and a degree of familiarity with the workings of the framework.

For these main reasons, we feel that designing a graphical user interface would be a great enhancement. It would allow users to easily alter simulation variables, without having to type each change, and pave the way to gradually adding more and more customization capabilities available through an efficient manner.

From another point of view, modelling of the phase change region of phase change materials involved the *apparent heat capacity* method. This method results in a series of extra computations involving big matrices and a consequent respectable overhead in the simulation time needed for 3D-ICE. As previously stated, the implementation we developed represents a trade-off between development time and performance. However, we are aware that this implementation is far from optimal and feel that optimizing this extra capability would further enhance the usefulness of the framework. It quite undisputable that simulation times are a very important factor in research.

In addition, a similar trade-off resulted in a number of statistical data being exported to text files for each simulation. While this implementation facilitates parsing and further data processing, it might not be optimal. A possible and more flexible approach would be to store such data in a database and accordingly retrieve information as needed.

Regarding phase change materials, this work focused mainly on researching the primary heat path in a chip (from the active components, through the copper spreader to the PCM layer and to the ambient). However, it might be of particular interest to properly model the secondary heat path (through the PCB to the main board), and research the effect of adding phase change materials of suitable characteristics along this heat path.

Another interesting idea would be to research the effect of using multiple stacked PCM layers of different melting points, or PCM layers with different melting points in specific regions, for example, on top of the hottest regions of each core. Let it be noted, that each of these cases is already supported by the framework.

References

[1] G. E. Moore, “Cramming more components into integrated circuit. Electronics”, 38(8), 1965

[2] R. Dennard, et al., “Design of ion-implanted MOSFETs with very small physical dimensions,” IEEE Journal of Solid State Circuits, vol. SC-9, no. 5, pp. 256-268, Oct. 1974

[3] Muhammad Shafique , Siddharth Garg , Jörg Henkel , Diana Marculescu, “The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives”, Proceedings of the 51st Annual Design Automation Conference, p.1-6, June 01-05, 2014, San Francisco, CA, USA

[4] Hadi Esmaeilzadeh , Emily Blem , Renée St. Amant , Karthikeyan Sankaralingam , Doug Burger, “Power challenges may end the multicore era”, Communications of the ACM, v.56 n.2, February 2013

[5] D. Geer, "Industry t D. Geer, "Industry trends: Chip makers turn to multicore processors", Computer, vol. 38, no. 5, pp. 11-13, 2005

[6] G. Blake, R. Dreslinski, and T. Mudge, "A Survey of Multicore Processors," IEEE Signal Processing Magazine, Vol. 26, No. 6, pp. 26-37,2009

[7] Krste Asanovic , Rastislav Bodik , James Demmel , Tony Keaveny , Kurt Keutzer , John Kubiatowicz , Nelson Morgan , David Patterson , Koushik Sen , John Wawrzynek , David Wessel , Katherine Yelick, “A view of the parallel computing landscape”, Communications of the ACM, v.52 n.10, October 2009

[8] Fulya Kaplan, and Ayse Kivilcim Coskun, “Adaptive sprinting: How to get the most out of Phase Change based passive cooling”, ISLPED, page 37-42. IEEE, (2015)

[9] Fulya Kaplan, "Modeling and analysis of phase change materials for efficient thermal management", 32nd International Conference on Computer Design, 2014

[10] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch and Milo M. K. Martin, “Computational Sprinting on a Hardware/Software Testbed “, In the Proceedings of the 18th Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), March 2013

[11] A. Raghavan et al., "Utilizing dark silicon to save energy with computational sprinting", Micro, IEEE, vol. 33 , no. 5, pp. 20-28, 2013

[12] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch and Milo M. K. Martin, "Computational Sprinting", In the Proceedings of the 18th Symposium on High Performance Computer Architecture (HPCA), February 2012

[13] https://en.wikipedia.org/wiki/Latent_heat#Specific_latent_heat

[14] https://en.wikipedia.org/wiki/Thermal_conductivity

[15] Andrea Tilli , Andrea Bartolini , Matteo Cacciari , Luca Benini, "Guaranteed Computational Resprinting via Model-Predictive Control", ACM Transactions on Embedded Computing Systems (TECS), v.14 n.3, May 2015

[16] Elon Bauer, Joseph Carlos, "Thermal Management Using PCM-Based Heatsinks"

[17] Sniper Power Simulator, http://snipersim.org/

[18] Sheng Li , Jung Ho Ahn , Richard D. Strong , Jay B. Brockman , Dean M. Tullsen , Norman P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures", Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, December 12-16, 2009, New York, New York

[19] A Sridhar, A Vincenzi, M Ruggiero, T Brunschwiler, D Atienza, "3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling", Proceedings of the 2010 International Conference on Computer-Aided Design (ICCAD 2010), San Jose, CA, USA, November 7-11 2010

[20] A Sridhar, A Vincenzi, M Ruggiero, T Brunschwiler, D Atienza, "Compact transient thermal model for 3D ICs with liquid cooling via enhanced heat transfer cavity geometries", Proceedings of the 16th International Workshop on Thermal Investigations of ICs and Systems (THERMINIC'10), Barcelona, Spain, 6-8 October, 2010

[21] Python Software Foundation. Python Language Reference, version 2.7. Available at https://www.python.org

[22] http://zeromq.org/

[23] J. Lienhard-IV and J. Lienhard-V, "A heat transfer textbook", Cambridge, Massachusetts: Phlogiston Press,2006

[24] MATLAB and Statistics Toolbox Release R2014a, The MathWorks, Inc., Natick, Massachusetts, United States

[25] G. Faust, R. Zhang, K. Skadron, M.R. Stan, and B. Meyer. "ArchFP: Rapid Prototyping of pre-RTL Floorplans." In Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Oct. 2012

[26] S. Memik , R. Mukherjee , M. Ni and J. Long, "Optimizing thermal sensor allocation for microprocessors", IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 3, pp. 516-527, 2008

[27] . Florea, C. R. Buduleci, R. Chis, A. Gellert, L. Vintan, "Enhancing the Sniper Simulator with Thermal Measurement", Proceedings of The 18-th International Conference on System Theory, Control and Computing, Sinaia (Romania), October 17-19, 2014 (submitted).

[28] Advanced Materials for Thermal Management of Electronic Packaging, Xingcun Colin Tong

[29] Area Array Interconnection Handbook, Karl J. Puttlitz, Paul A. T