3.3. Изследване на SMT архитектури

С увеличаване на възможността за по-голям брой транзистори в VLSI технологията на един чип се търсят други хардуерни решения, които да експлоатират ефективно големия брой транзистори. Основният недостатък на ССПА е, че те използват паралелизъм на ниво инструкции. Естественото решение е да се раздели програмата на нишки или подпрограми, които вътре в тях се използва паралелизъм на ниво инструкции(ILP).

Figure 1: Empty issue slots can be defined as either vertical waste or horizontal waste. Vertical waste is introduced when the processor issues no instructions in a cycle, horizontal waste when not all issue slots can be filled in a cycle. Superscalar execution (as opposed to single-issue execution) both introduces horizontal waste and increases the amount of vertical waste.

Мултискаларен процесор

Мултискаларният процесор предложен от Сухи[44] полага основата на т.нар. спекулативен многонишков подход. Той използва нова агресивна парадигма за извличане на голямо количество ILP от програмни написани на високо ниво. Програмата се разделя на задачи (нишки) чрез комбинация от софтуер и хардуер. Задачите се разпределят върху отделните процесорни елементи. Всеки един от тях извлича и изпълнява инструкциите от неговата задача(нишка). Тази публикация е основополагаща в по-нататъшните изследвания върху паралелизма на ниво нишка.

Програмата се представя като граф на управляващия поток (Control Flow Graph), където основните блокове са възлите, а дъгите представят управляващия блок от един възел към друг. За да се постигне висока производителност процесорът трябва да изпълнява този граф с високо ниво на паралелизъм на ниво нишки(TLP). Логиката за предсказване на прехода със спекулативно изпълнение е техниката, която се използва за повишаване на нивото на TLP. Основният проблем е, че трябва да се спазва програмната последователност. В мултискаларния модел на изпълнение CFG е разделен на задачи(нишки). Мултискаларният процесор преминава CFG спекулативно. Задачите(нишките) се разпределят върху отделните процесорни елементи и се изпълняват паралелно. На това ниво концепцията изглежда проста, обаче ключовия елемент, за да работи ефективно е правилното откриване на междунишкови даннови зависимости. Може да се приеме консервативен или

агресивен подход за преодоляването на тези зависимости. Консервативният подход е да се изчака докато е сигурно, че дадената инструкция за четене от паметта ще зареди коректна стойност. Този подход предполага задържане на тези инструкции в нишката докато всички предишни нишки са завършили изпълнението на инструкциите за записи, което очевидно ще доведе до последователно изпълнение на програмата. Агресивният подход изпълнява инструкциите за четене агресивно, но с надеждата че предишните нишки няма да записват в тази клетка. Това изисква въвеждането на допълнителен хардуер, които да следи за възникването на неправилно предсказване и за възстановяването на програмната последователност. След тази публикация се правят изследвания от различни университети, разглеждащи проблема с автоматичното разпаралеляване на програмата(софтуерно/хардуерно), организацията на мултипроцесора, кеш-кохерентните схеми, поддържащи спекулативното изпълнение, мащабируемост на процесорните елементи в процесорната архитектура.

Спекулативен мултипроцесори

В продължение на много години процесорните архитекти обръщат голямо внимане на тази архитектура и предлагат различни варианти на изпълнение на multithreading [37,44,102]. Простотата при проектирането на CMP подхода, не са позволяват намаляването на тактовата честота във всеки процесорен елемент, но и позволява по-ефективно използване на ресурсите. CMP позволява всеки процесорен елемент да има по-голяма дълбочина на извличане на инструкциите в сравнение с ССПА[76]. Акуотън[76] представят как CMP с осем 2-issue суперскаларни процесорни елементи може да работи като един 12—issue суперскаларен процесор. От софтуерна гледна точна CMP подход е идеална платформа за многонишкови приложения. За съжаление, тази платформа не е напълно достъпна, защото не дава добра производителност при стартиране на последователни (single thread) приложения. С компилаторите за разпаралеляване на кода [138] този подход е подходящ само за ограничен клас от приложения и най-вече за числови изчисления. Даже и при тези приложения, компилаторът е много опростен и

приема наличието на вътрешни нишки, когато той може да гарантира независимости между инструкциите в тях.

Simultaneous Multithreading

SMT [31] подхода и неговите разновидности са обект на интензивно изучаване. Най-голямо внимание на този подход му е обърна Тулсън[27]. Той позволява много нишки да се съревновават за разпределението на дадените процесорни ресурси на всеки такт. Едно от ключовите предимства, когато изпълняваме паралелни приложения е неговата способност да използва двата паралелизма (ILP и TLP) едновременно, като позволява няколко нишки да споделят ресурсите едновременно (simulteneous). Когато програмата има не голям TLP, всичките ресурси на процесора ще бъдат предоставени на тази нишка, а когато съществува по-голям TLP (т.е. има няколко нишки) – той ще компенсира липсата на ILP във всяка нишка. SMT процесорът може уникално да експлоатира, които и да е тип паралелизъм, по този начин използва функционалните устройства много по-ефективно и постига по-голяма пропускателна способност(throughput). Предишни изследвания са правени като е използвано мултипрограмно натоварване(workload) с цел оценка потенциала на SMT архитектури. Едно от основните предимства на SMT е това, че той може да бъде получен от SS п-р с малки изменения в архитектурата и както показват изследванията това води до повишаване на производителността спрямо wide issue SS п-р. Мултипрограмното натоварване доставя много TLP, защото всяка нишка принадлежи на изцяло различно приложение. Различните програми имат различни изисквания към SMT, отколкото мултипрограмното натоварване.

SMT практиката – hyperthreading

Тази парадигма е въведена от Интел и за първи път е приложена в процесора Xeon MP, през 2002г. [129]. Тя позволява изпълнението на няколко нишки едновременно, които си споделят процесорните ресурси. Физическият процесор Xeon MP има два логически процесора, които споделят процесорните ресурси. Операционната система и приложенията виждат два процесора и могат да разпределят задачите между тях, както е в случаите на двупроцесорна архитектура.

Hyper-Threading е базиран на принципа в даден момент от време част от процесорните ресурси се използват за изпълнение от една нишка. Другата част

от неизползваемите ресурси могат да се предоставят за достъп на друга нишка. Идеята е една нишка да натоварва процесора така както е при еднопроцесорните архитектури. Затова две-ядрения процесор има два режима на работа: Single tasking (ST) и Multi-Tasking(MT). При първият режим на работа само единия процесор е активен, а другият е спрян с HALT инструкцията. Когато се появи втора ниша вторият процесор се включва и физическия процесор сменя режима на работа в MT.

Всеки един от двата логически процесора имат свое архитектурно състояние, което включва състояние на регистрите – регистрите с общо приложение, управляващите регистри, контролер на прекъсванията (APIC). Освен това логическите процесори имат свои собствено множество регистри и контролер на прекъсванията. Имат RAT таблица за 8 регистъра с общо предназначение IA-32 и споделят регистрите на физическия процесор (по една таблица за всеки логически процесор).

Когато две нишки се изпълняват има два указателя за следващи инструкции. Голяма част от инструкциите са в трейс кеша, където се съхраняват декодирани и на всеки такт логическите процесори ги извличат. Буферът ITLB е дублиран и доставя инструкции за двете нишки. Фазата за декодиране е обща за двата логически процесора и когато две нишки се нуждаят от декодирани инструкции, то тази фаза ги обслужва едновременно.

Техники за ускоряване производителността на съвременните процесори

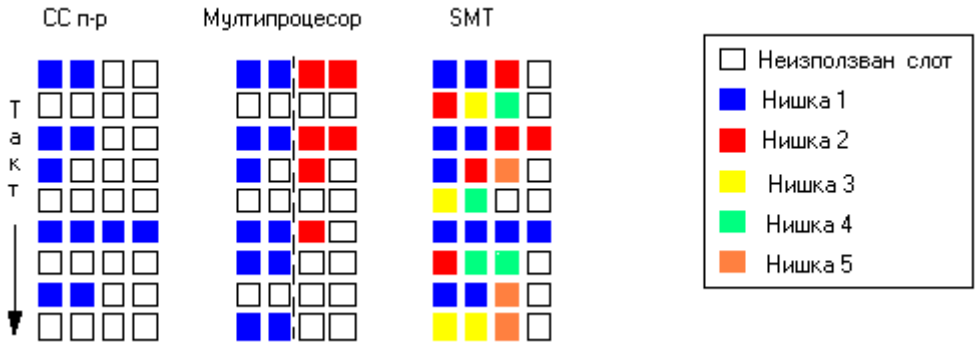
За да се постигне висока производителност съвременните компютърни архитектури използват две форми на паралелизъм – паралелизъм на ниво инструкции (ILP) и паралелизъм на ниво нишки (TLP). Въпреки че, те имат различна гранулярност те са фундаментално идентични, защото откриват независими инструкции, които могат да бъдат изпълнени паралелно и по този начин използват паралелния HW. Така наречените wide issue суперскаларните процесори експлоатират ILP чрез изпълнение на множество инструкции от еднонишкова програма за всеки такт. Мултипроцесорите откриват TLP, чрез паралелно изпълнение на различни нишки върху няколко процесора. Но нито един от двата подхода не е способен да се адаптира към динамичното изменение на нивата на ILP и TLP, защото HW комбинация е насочена твърдо към един от двата типа паралелизъм. Недостатък при мултипроцесорите е, че ако имаме недостатъчно TLP в програмата, някои от процесорите ще останат неизползваеми. От друга страна, суперскаларният процесор изпълнява една нишка и ако имаме недостатъчно ILP, голяма част от ресурсите ще остане

неизползвани – особено тези, които са свързани с HW за едновременно подаване.

Simultaneous Multi-Threading (SMT) или т.нар. HyperThreading заимства предимствата на двата подхода, като позволява много нишки да се съревновават за разпределението на дадените процесорни ресурси всеки цикъл. Едно от ключовите предимства, когато изпълняваме паралелни приложения е способността на SMT процесора да използва двата паралелизма едновременно, като позволява няколко нишки да споделят процесорните ресурси едновременно (*simultaneous*). Когато програмата има само една нишка, т.е. липсва TLP, всичките ресурси на процесора ще бъдат предоставени на тази нишка, а когато съществува по-голям TLP (има няколко нишки) – той ще компенсира липсата на ILP във всяка нишка. SMT процесорът може уникално да експлоатира които и да е тип паралелизъм, по този начин използва функционалните устройства много по-ефективно и постига по-голяма пропускателна способност, следователно значително ускоряване на програмата.

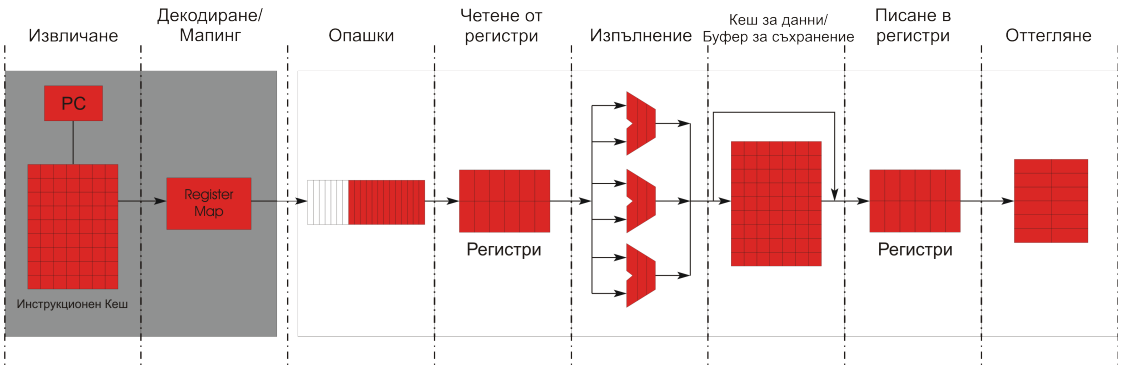
Изследванията в дисертацията са насочени към паралелно изпълнение при SMT архитектура. Предишни изследвания са правени като е използвано мултипрограмно натоварване (*workload*) с цел оценка потенциала на SMT архитектури. Едно от основните предимства на SMT е това, че той може да бъде получен от SS процесор с малки изменения в архитектурата и както показват изследванията това води до повишаване на производителността спрямо *wide issue* суперскаларен процесор. Мултипрограмното натоварване доставя много TLP, защото всяка нишка принадлежи на изцяло различно приложение.

Както вече споменах обект на мои изследвания са паралелните програми, т.е. една програма е разделена на нишки, които се синхронизират и споделят данните. Тези програми имат по-различни изисквания към SMT, отколкото мултипрограмното натоварване. В частност сравнявам SMT с CMP – това са small scale on chip multiprocessors и оценявам способността да откриват ILP и TLP(фиг.3.21).



фиг.3.21. Различни подходи

При суперскаларните процесори през по-голяма част от времето, повечето ресурси на системата не се използват, тъй като програмата има ниско ниво ILP. При мултипроцесорите (CMP) могат едновременно да се изпълнят няколко нишки върху различни процесори, но производителността им е ограничена от фиксираното разделяне между процесорите. От фиг.3.21 се вижда, че има недостатъчен TLP в програмата, т.е. някои процесори ще останат неизползвани. Друг недостатък при мултипроцесорите е използването на процесори с ниско ниво на ILP. SMT процесорът преодолява недостатъците на предишните два подхода, като има способността да открива едновременно ILP и TLP. В случаят, при SMT може да се подават четири инструкции от четири различни нишки едновременно. Динамично се разпределят ресурсите между нишките и когато има само една нишка всички ресурси се предоставят на нея, в противен случай високото ниво на TLP ще се компенсира липсата на ILP във всяка една нишка и по този начин няма да имаме празни слотове във базовия суперскаларен процесор.



фиг.3.22. Основна Архитектура на Суперскаларен процесор