ВЪВЕДЕНИЕ

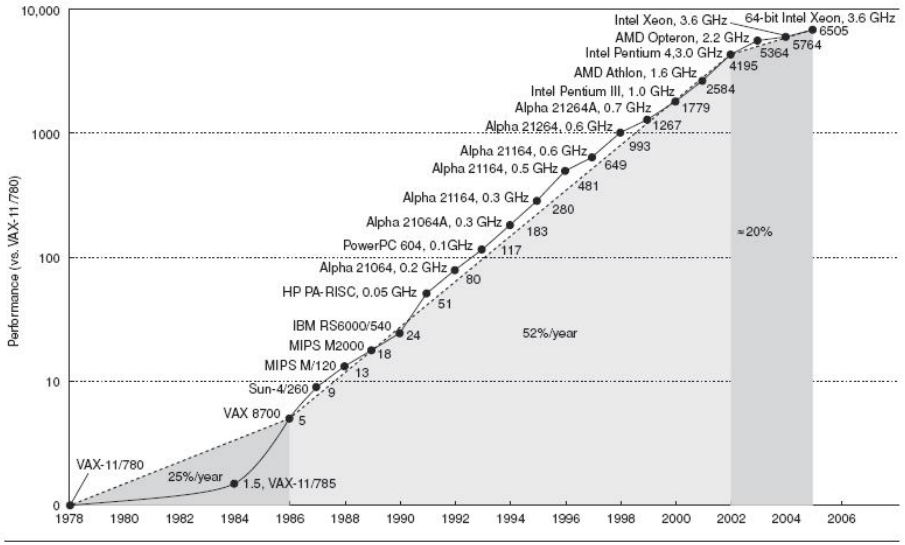
Водещ методологичен принцип в науката е изследванията да се насочват към основното, същественото противоречие на предметната област или конкретна нейна подобласт. Разбира се, на първо място се предполага да се открие водещото противоречие, чието разрешаване води до издигането на предметната област на следващо, по-високо ниво. При това не трябва да се забравя и вътрешната динамика и логиката на развитието - разрешаването на едни противоречия води неизбежно до появата на други.

Последните 50 години от историята на компютърната индустрия са потвърждение на тази водеща научна теза. Например, в областта на компютърните системи и архитектури[1], като част от компютърната индустрия, са известни множество противоречия, някои от които са:

- противоречието между апаратната и програмната реализация;
- противоречието между езиците за програмиране от едно ниво, както и между езиците за програмиране от различно ниво;
- противоречието между различните операционни системи;
- противоречието между семантиката на езиците от високо ниво и поддържаната от физическата машина семантика;
- противоречието между необходимостта от програмна съвместимост на ниво изпълним код с предходните поколения и необходимостта от усъвършенстване на архитектурата;
- противоречието между масовото разпространение на 32-bit софтуер и необходимостта от преход към *64*-bit архитектури;
- противоречието между различните нива на поддръжка на паралелизма;
- противоречието между последователното мислене, последователните алгоритми, последователните езици за програмиране и съвременните архитектури с явен паралелизъм.

Някои от тези противоречия са преодоляни, други не. Съответно, някои от противоречията са временни, други - вътрешноприсъщи и

1 Когато говорим за компютърната архитектура, тя трябва да се разглежда в триединството "*архитектура - езици за програмиране - операционна система*".



Фиг. B1. Графика на развитието на производителността

постоянни. Някои са съществени в даден период от време, други - несъществени в този период.

От посочените противоречия несъществени в момента са противоречията между различните езици за програмиране, между различните операционни системи[2].

Постоянно е противоречието между апаратната и програмната реализация, в частност - между различните нива на поддръжка на паралелизма[3].

Анализът на развитието на компютърните системи и архитектури през последните няколко десетилетия позволява да се ориентираме в множеството противоречия и да отделим най-съществените от тях за съвременния етап: между последователното мислене, последователните алгоритми, последователните езици за програмиране и съвременните архитектури с явен паралелизъм. Противоречие, което може да се определи за третата особена точка или *третата сингулярност* в графиката на развитието на производителността.

В [29] пионерите на *RISC* архитектурите Хенеси и Патерсън обръщат внимание на двете *особени точки* в графиката на развитието на производителността, показана на фиг. B1. Тези особени точки са свързани с промяната на наклона на графиката.

---

2 Не степента на експониране на даден въпрос пред масовия потребител определя неговото значение.

3 Постоянният характер на дадено противоречие в известна степен го притъпява.

Първата особена точка в средата на 80-те години се дължи на навлизането на *RISC* архитектурите и използвания от тях локален паралелизъм на ниво инструкции (*ILP*).

Втората особена точка, около 2003 година, отговаря на изчерпването на локалния паралелизъм и навлизането на структурните методи за повишаване на производителността – многоядрените архитектури, които са пример за *явен* глобален паралелизъм[4].

В този контекст е полезно да се проследи графиката на развитието на производителността, привеждана в последните няколко издания на фундаменталния учебник на Хенеси и Патерсън по компютърни архитектури [28, 29, 30].

Изводите на Хенеси и Патерсън се потвърждават и от други, независими източници. Например в [41, 42] се посочва близка по вид и стойности графика. Прави се извод за изчерпването на прираста в производителността, който може да бъде получен със средствата на локалния паралелизъм на ниво инструкции[5]. *Конвейеризацията* на инструкциите, като основен елемент на *ILP*, се заключава в извличането на съдържащия се в последователните програми скрит паралелизъм. Този паралелизъм обаче е ограничен. И посочваните емпирични резултати доказват достигането на тези ограничения.

Намаляването на ефекта от *ILP* е практическо следствие на *принципа за намаляващия ефект от оптимизациите* (the low of diminishing returns). Този принцип отчита намаляването на ефекта от една и съща оптимизация в рамките на ограничена област.

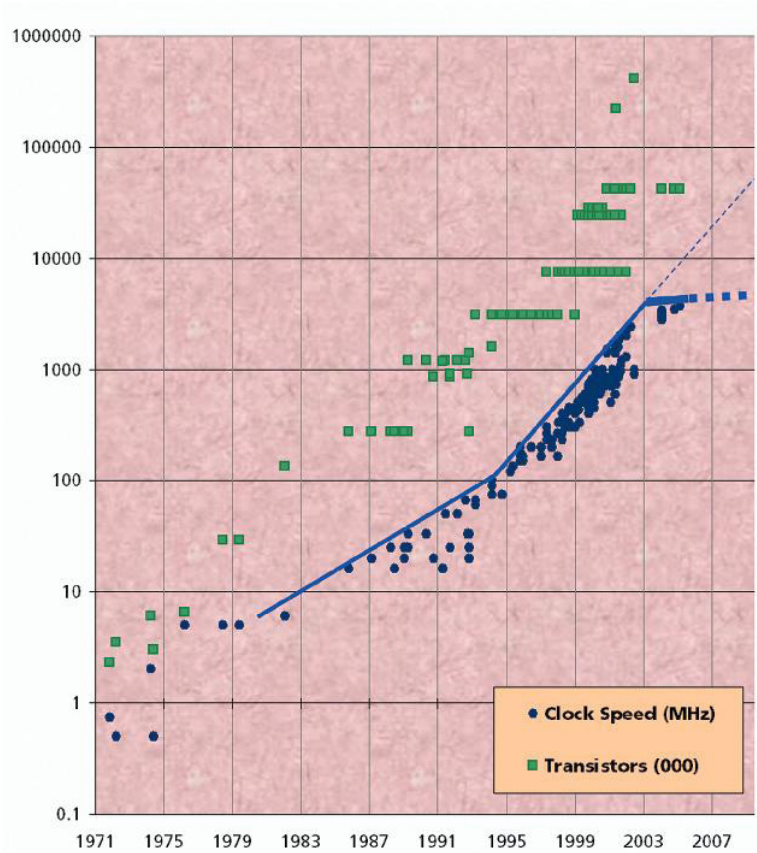
Някои от основните източници на ограниченията на *ILP* се разглеждат например в [42]. Броят *n* на динамично извлечените за паралелно изпълнение инструкции от суперскаларните архитектури се ограничава от сложността на необходимата апаратна логика. Доказва се, че тази сложност е $O(n^2)$. Подобна е ситуацията и при *VLIW* архитектурите, където този паралелизъм се извлича статично, на етапа на транслацията.

От друга страна, дълбочината на конвейерите на инструкциите, т.е. броя на стъпалата им, е в диапазона от 10 до 20. Увеличаването на този брой над 20 води до недопустимо нарастване на загубите от преходните процеси в конвейерните регистри, разделящи стъпалата. Нещо повече, опитът от използването на *NetBurst* архитектурата при *Pentium IV* (една от последователните машини с най-дълбоки конвейери) показва практическата невъзможност да се натоварят всички стъпала. Това е следствие на споменатото фундаментално

---

4 За разлика от локалния пararлелизъм, който по правило е *неявен*.

5 Изчерпването на *ILP* не означава, че този метод спира да се използва. Напротив, той служи като база за следващите степени на развитие и поддръжката му се счита за подразбираща се.



Фиг. B2. Графика на изменението на тактовата честота на процесорите

ограничение на *ILP* – ограничения обем на скрития в рамките на последователните алгоритми паралелизъм.

На фиг. B2 е показана графиката на изменението на тактовата честота на процесорите през последните 30 години [46]. И тук се забелязват двете особени точки от фиг. B1. Първата - около средата на 90-те години, се обяснява с рязкото намаляване на периода на машинния цикъл, следствие *конвейеризацията* на инструкциите[6].

И докато първата особена точка в двете графики е изместена във времето, заради особеностите на влиянието на конвейеризацията върху производителността и тактовата честота, то втората особена точка съвпада по време и на двете фигури. По отношение на тактовата честота, тази особена точка се обяснява с достигането

6 Конвейеризацията на инструкциите е основен метод за достигане целта на *RISC* архитектурите - максимално висока производителност. Така, с навлизането на този клас архитектури се разпространява и използването на конвейеризацията на инструкциите.

на *фундаментални физически ограничения* – крайната скорост на разпространение на електрическия сигнал (“*светлинната бариера*“) и ограничението в максималното количество отделяна топлина от единица площ на полупроводниковия кристал (“*топлинната бариера*“).

Ако например тактовата честота на процесора f_c е *5* GHz, то периодът на машинния цикъл t_c ще бъде *0,2* ns. Скоростта на разпространение на светлината във вакуум *c* е *300 000* km/sec, т.е. *30* cm/ns. Оттук, максималното разстояние, което може да се измине за периода на един машинен цикъл, е 6 cm. Разбира се, реалното разстояние, което може да се измине от електрическите токоносители в полупроводниковия кристал, е по-малко.

От друга страна, увеличаването на броя на транзисторите в интегралните схеми води до нарастване на утечните токове, а оттам – до по-голяма консумация. За да се избегне този ефект, с нарастването на броя на транзисторите не се увеличава тактовата честота, а паралелизма на структурата [30].

Консумираната мощност *P* на *CMOS VLSI* схемите е функция на динамичния капацитет *C*, захранващото напрежение *V* и тактовата честота *F*

$$P = C \times V^2 \times F\,, \qquad \text{(B.1)}$$

където динамичният капацитет е правопропорционален на броя на транзисторите в интегралната схема.

Предпоставка за развитието на съвременните паралелни архитектури е високата степен на интеграция на свръхголемите интегрални схеми (*UVLSI*, СГИС). Едновременно с това, нарастващата степен на паралелизъм е предизвикана от необходимостта да се преодолеят физическите ограничения върху тактовата честота и консумираната електрическа енергия, от ограниченията в размера на транзисторите, от ограниченията на локалния паралелизъм.

Преминаването от последователен към паралелен изчислителен модел изисква промяна и в методите и средствата на програмиране, в начина на мислене, което можем да определим като *третата сингулярност*.

+++

Работата е оформена в три главни раздела.

В *първия раздел* се разглеждат основните методи за генерация на случайни последователности. Изборът им е съобразен с евентуалното им приложение в следващите етапи от изследванията и не претендират за изчерпателност.

Разглежда се един от основните математически модели за описание на паралелни системи с разпределена памет е *CSP* [16, 31]. Той се базира на понятието *процес* като базова активна единица[7]. Всеки процес се изпълнява в локална защитена среда (обособен апаратен компютърен *възел*) и единственият начин за взаимодействие с останалите процеси е обменът на съобщения по физически *канали*.

Освен абстрактен модел на паралелните изчисления, *CSP* представлява и език за паралелно програмиране. В него са дефинирани оператори[8] за паралелна и последователна композиция на процеси, за недетерминиран избор, за взаимодействие между процесите и др.

Интересна особеност на *CSP* е наличието на физически реализации на описваната от него абстрактна паралелна машина. Първата реализация на *CSP*-машина е паралелната платформа Транспютър/OCCAM.

Паралелната платформа *XCORE/XC* е съвременната реализация на абстрактната *CSP*-машина, т.е. паралелна изпълнителна среда. Ядрото *XCORE* от фамилията *XS1* на фирмата *XMOS* поддържа глобален паралелизъм на микроархитектурно ниво: създаване/унищожаване, изпълнение, синхронизация и взаимодействие на паралелни процеси.

Фамилията *XS1* се програмира на езика *XC* - разширение на езика *C* с конструкции за явно управление на паралелизма, взаимодействието, събитията, вход-изхода. Това разширение представлява програмен интерфейс към поддържаните на микроархитектурно ниво механизми на глобалния структурен паралелизъм.

Прави се преглед на използваната паралелна изпълнителна среда.

Вторият раздел е посветен на реализацията и изследването на група генератори. Определен интерес представлява реализацията на генератор на действително-случайни последователности чрез наличния в изпълнителната среда ентропиен източник. Изследва се наличния недетерминизъм в оператора *select*, който в езика *XC* е аналог на алтернативната команда на *CSP*[9]. Предлага се метод за вграждане на недетерминизъм в оператора *select*, с оглед подо-

---

7 На абстрактно ниво *процесът* се определя като основната единица за декомпозиране активността на системата.

8 Допуска се операторите на *CSP* да се разглеждат също и като команди.

9 Алтернативната команда е известна също като команда за недетерминиран избор.

бряване адекватността му на алтернативната команда.

Предложен е вариант на монобитния тест за проверка на генерираната случайна последователност. Тестът протича в два етапа. През първия етап се натрупва първоначалната извадка и началната статистика. По време на втория етап се извършва инкрементално обновяване на статистиката в реално време. Полезна е възможността за изпълнение на монобитния тест с χ^2 критерий в реално време. Предполага се целочислена аритметика и използването на *MAC* инструкцията. Приведени са някои от получените експериментални статистически данни.

В *третият раздел* се разглеждат основните моменти от експерименталната оценка на предложените решения.

Работата се придружава от паралелния код на разработените генератори. Това позволява резултатите да се проверят от независими изследователски групи, както и да се използват в изследвания, предполагащи наличието на източници на случайни последователности в паралелна среда.

Не на последно място, разгледаният материал може да помогне на мислещия компютърен специалист в преодоляването на *третата сингулярност*.