1.1. ПАРАЛЕЛНА *SMT/TLP* ИЗПЪЛНИТЕЛНА СРЕДА *XCORE/XC*

Предпоставка за прехода от последователния фонноймановкъм паралелния изчислителен модел са физическите ограничения за увеличаването на производителността, по-конкретно светлинната и топлинната бариера, достигнати около 2003 година от суперскаларните машини.

Пионерите на *RISC* архитектурите Хенеси и Патерсън обръщат внимание на втората особена точка в графиката на развитието на производителността, настъпила в този период. Тази особена точка, илюстрирана на фиг. В.1 с реални данни, отговаря на изчерпването на локалния паралелизъм (*ILP*) и навлизането на структурните методи за повишаване на производителността – главно многоядрените архитектури, които са пример за глобален паралелизъм на ниво нишки (*TLP*) и на ниво данни (*DLP*). Това налага все по-остро намирането на обща методология за паралелно програмиране.

Самият принцип на работа на класическите архитектури с общо предназначение обаче се явява пречка – тези архитектури са последователни по определение. Изражение на това е практическото отсъствие при тях на апаратна поддръжка на глобалния паралелизъм – създаването и унищожаването, изпълнението, взаимодействието и синхронизацията на процесите по правило се осигуряват от операционната система, а не от самата архитектура.

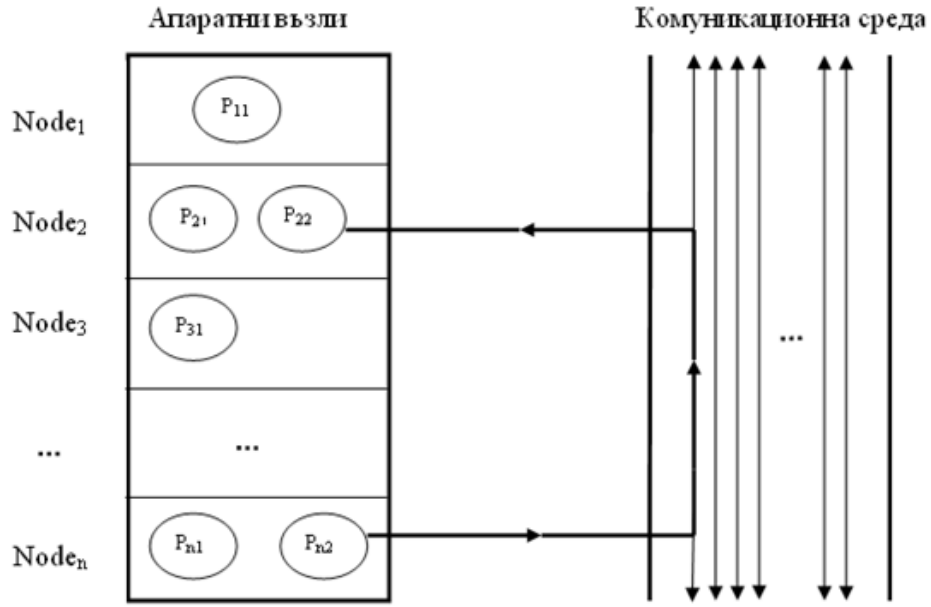
Алтернатива представляват реализациите на *CSP* машината, каквито са транспютрите и езика *OCCAM*, ядрото *XCORE* и езика *XC*.

CSP (*Communicating Sequential Processes*, Взаимодействащи последователни процеси) е предложен от Чарлз Хоар [Б.2, Б.5]. Моделът се базира на понятието *процес* като базова активна единица. Процесът сам по себе си може да бъде съставен, т.е. да представлява съвкупност от други процеси.

Структурата на всеки процес може да се определи с рекурсивния израз:

P : = < линеен участък > < комуникация > *P*

Следователно, в тялото на процеса съществуват два основни участъка – линеен участък, представляващ последователност от оператори и участък, в който процесът взаимодейства с останалите



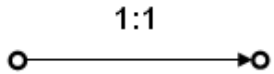
Фиг. 1.1. Структура на *CSP* машина

процеси в системата.

Изборът на тази структура на процесите е естествено следствие от разширяването на последователните машини и последователните езици за програмиране със средства за комуникация, при което се отчитат особеностите на разпределения модел памет. В съответствие с този модел се определя абстрактна *CSP* машина, която се състои от множество апаратни компютърни възли[1] и комуникационна среда за връзка между възлите (фиг. 1.1).

Всеки един процес се изпълнява в локална защитена среда (апаратния възел). Единственият начин за взаимодействие с останалите процеси е обменът на съобщения по свързващите ги комуникационни канали.

Обменът на съобщения, т.е. взаимодействието, се извършва чрез комуникационната среда, която представлява множество физически канали. Всеки такъв канал служи за двуточково еднопосочно взаимодействие (фиг. 1.2).



Фиг. 1.2. Двуточково еднопосочно взаимодействие

1 Апаратният възел (*Node*) е завършен компютър - със собствен процесор, собствена памет и интерфейсни схеми за връзка със средата.

Комуникацията е с двустранна синхронизация от типа рандеву (*rendezvous*). Това означава, че и двата процеса трябва да достигнат командата за взаимодействие, за да може то да се осъществи. Процесът, който е изпреварил другия се задържа (*блокира*[2]) толкова време колкото е необходимо. Подобен избор се обосновава с необходимостта да се използват максимално опростени канали - еднопосочни, без буферизация и без контрол на потока (*flow control*).

Дефинирани са няколко оператора на *CSP*, които се разглеждат и като команди.

Операторът за *паралелна композиция* (паралелната команда) има следния синтаксис

$P = \{ P_1 \,||\, P_2 \,||\, \ldots \,||\, P_n \}$

Паралелната команда определя съставния процес *P* като съвкупност от *n* на брой паралелни процеса, които се стартират и изпълняват едновременно. Командата приключва с приключването на най-продължителния от съставящите я процеси.

Операторът за *последователна композиция* (последователната команда) има следния синтаксис

$P = P_1; P_2; \ldots; P_n$

Последователната команда служи за конструиране на съставния процес *P*, състоящ се от *n* последователно изпълнявани процеса.

Операторът за *алтернативен избор* има следния синтаксис

$P = \{G_1 \rightarrow P_1 \,\square\, G_2 \rightarrow P_2 \,\square\, \ldots \,\square\, G_n \rightarrow P_n\}$

Този оператор е подобен на класическия оператор за избор *if/else*. Но за разлика от него, поддържа недетерминизъм, откъдето е известен и като оператор за *недетерминиран избор*. Недетерминизмът е вътрешно присъщ на всяка една паралелна система, което фактически е отчетено с въвеждането на оператора за алтернативен избор.

Двойката $G_i \rightarrow P_i$ се нарича *i-та* алтернатива, където защитата (*guard*) G_i се разглежда като булева променлива[3], а P_i е подчинения на G_i процес. P_i може да се изпълни единствено, ако защитата му G_i е истина.

Изпълнението на оператора за алтернативен избор протича по следната схема: *Едновременно* се проверяват всички защити. Ако

2 Даден процес се разглежда като *блокиран*, ако е изведен от опашката на диспечера на операционната система (опашката на готовите задачи). Механизмът на блокирането на процесите служи за намаляване на системните загуби - избягват се циклите на изчакване, които товарят непроизводително процесора.

3 Най-простият вариант на *защита* (guard) е булева променлива. Често обаче се среща и значително по-сложния вариант, когато ролята на защита изпълнява *команда за взаимодействие*. Затова е редно защитата да се разглежда като предикат, който връща булев резултат.

нито една от защитите не е истина, операторът се прекратява. Ако само една от защитите G_i е истина, се изпълнява нейния подчинен процес P_i. Ако повече от една защита е истина, по *случаен закон* се избира една от тях и се преминава към изпълнение на подчинения й процес. Ключови са едновременната, а не последователната, проверка на защитите и случайният, а не детерминираният, закон за избор.

Алтернативната команда фактически е обобщение и съдържа като частен случай оператора за детерминиран избор *if/else* от последователните езици.

Операторът

* { < *тяло* >}

определя *цикличен процес*. За неговото управление се налага да се използват двата елементарни процеса *STOP* и *SKIP*. При достигане на *STOP* командата се прекратява, докато при *SKIP* се извършва преход в началото й.

Командите за *взаимодействие* са две – за изпращане на съобщение и за приемане на съобщение.

Командата за изпращане (*send*) на съобщението *msg* по канала *c* има вида

c ! *msg*

Съответно, командата за получаване (*receive*) на съобщението от канала *c* и записването му в променливата *var* е от вида

c ? *var*

И двете команди за взаимодействие използват директно именоване на каналите и променливите. Това дава възможност на всеки един процес да определи момента на взаимодействието и процеса, с който се взаимодейства.

Важна особеност на *CSP* е, че представлява едновременно математически модел, формална спецификация на паралелна система и език за паралелно програмиране. Нещо повече, съществуват реални апаратни системи, отговарящи на *CSP* машината и реални езици за паралелно програмиране, производни на *CSP* (фиг. 1.3).

Първият пример за практическо приложение на *CSP* са транспютърните паралелни системи и езикът за паралелно програмиране *OCCAM* [Б.1].

Транспютрите (TRANSistor comPUTER) представляват специализирани едночипови микрокомпютри, предназначени за изграждане на масово-паралелни системи (*MPP*). За разлика от универсалните процесори, например *Intel x86*, те осигуряват цялостна апаратна поддръжка на процеса, като базов активен

Фиг. 1.3. Връзка между абстрактната и реалната страна на *CSP*

обект, и превръщането му в елементарна единица работа[4]. Освен това притежават и вградени комуникационни средства – канали. Произвеждани са от английската фирма *InMOS*.

През 2005 е създадена фирмата *XMOS*. Един от основателите ѝ - Дейвид Мей (David May) е водещия архитект на транспютрите. Фирмата *XMOS* предлага в момента микропроцесорната фамилия *XS1*, която в своята основа е много близка до транспютрите и е следващата реализация на *CSP* машината.

На фиг. 1.4 са показани основните параметри на различни микропроцесори от тази фамилия [Б.6]: устройствата с ниска консумация *XS1-L1*, *XS1-SU1*, *XS1-L2* и високопроизводителния чип *XS1-G4*.

И четирите посочени разновидности на фамилията *XS1* съдържат едно или повече ядра от типа *XCORE*. Това ядро представлява *събитиен* (*event-driven*) *RISC* процесор със *силносвързан I/O*. Поддържа едновременното изпълнение на 8 нишки, което го причислява към *SMT/TLP* процесорите (фиг. 1.5). Използва се дребногранулирано многонишково изпълнение (*fine-grained MTP*), като във всеки машинен цикъл се извършва превключване на контекст на нишка, за разлика от едрогранулираното многонишково

4 Критерий за тежестта на дадена работа за процесора е отношението на времето за нейното изпълнение към времето за изпълнение на основните машинни инструкции (зареждане/запомняне, аритметико-логически и др.). Елементарна е такава единица, за която това отношение е в диапазона от 1 до не повече от 10.

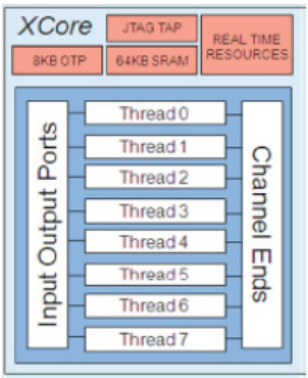
Device	XS1-L1	XS1-SU1	XS1-L2	XS1-G4
XCores	1	1	2	4
Threads	8	8	16	32
MIPS (max)	500	700	1000	1600
SRAM (total)	64 KBytes	64 KBytes	128 KBytes	256 KBytes
OTP (total)	8 KBytes	8 KBytes	16 KBytes	32 KBytes
Supply I/O, core	3v3, 1v0	3v3 or 5v0, 3v3	3v3, 1v0	3v3, 1v0
Power consumption	15-200mW	<1-300mW	30-400mW	200-1200mW
Analog	-	USB, ADC	-	-
Packages (digital I/O)	QFP48 (28) QFP64 (36) QFP128 (64)	BGA96 (37)	QFN124 (84)	BGA144 (88) BGA512 (256)

Фиг. 1.4. Представители на фамилията *XS1*

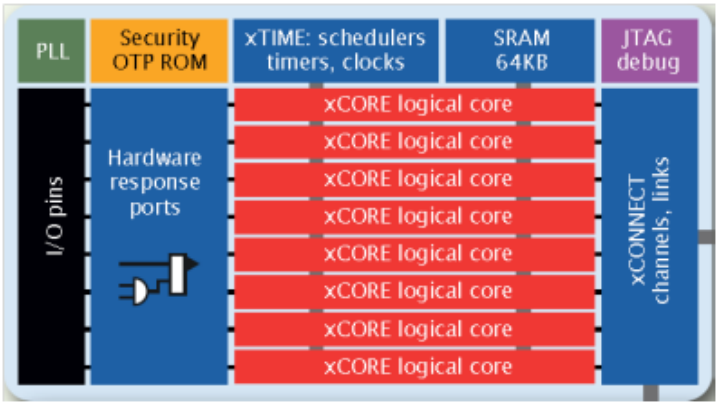
изпълнение (*coarse-grained SMT*, Simultaneous Multithreading) при *Intel x86* и *PowerPC*. *TLP* и взаимодействието между процеси се поддържат изцяло апаратно и не изискват *ОС*.

На фиг. 1.5 са показани основните ресурси, влизащи в състава на ядрото *XCORE*: *SMT/TLP* апаратна среда, поддържаща едновременното изпълнение на 8 апаратни нишки; силносвързаната входно-изходна система; физическите канали за връзка; оперативна памет от типа *SRAM*; постоянна (*OTP*) памет; ресурси за работа в реално време; *JTAG* интерфейс за връзка с инструментален компютър.

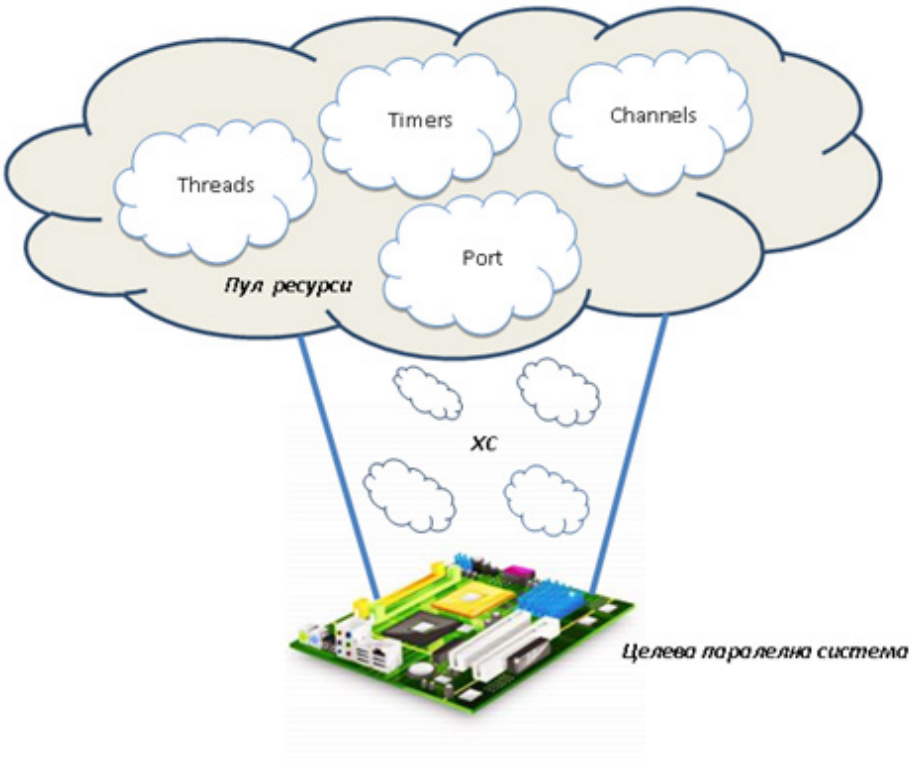
На фиг. 1.6 са показани промените в обозначенията, възприети в момента от фирмата *XMOS*. Физическото ядро вече се обозначава с термина *tile*. Докато вместо апаратна нишка се говори за *логическо ядро* (*logical core*). Тези промени са важни за конфигурационния файл *XN*, който съдържа описанието на апаратната конфигурация.



Фиг. 1. 5. Обща структура на ядрото *XCORE* с архитектура *XS1*



Фиг. 1. 6. Обща структура на ядрото *XCORE* с новите обозначения, приети от фирмата *XMOS*



Фиг. 1. 7. Концепция *софтуерно-дефиниран силиций*

Фамилията *XS1* се програмира на езика *XC* [B.11, B.15, B.17], който е разширение на езика *C* с конструкции за паралелно програмиране – явно управление на паралелизма, взаимодействието, събитията и вход-изхода. Благодарение на апаратната поддръжка на *CSP* модела, операторите на *XC* се транслират в къси *RISC* инструкции, като се избягват служебните загуби, характерни за *ОС*.

Както е показано на фиг. 1.3, езикът за паралелно програмиране *XC* е реализация на *CSP*. Може да се проследи например семантическата връзка между конструкцията *par* на *XC* и паралелната команда на *CSP*; между оператора *select* и алтернативната команда; между операторите за въвеждане *:>* и извеждане *<:* и командите за изпращане *!* и получаване *?* на съобщения и т.н.

Паралелният език *XC* е в основата и на концепцията *софтуерно-дефиниран силиций* (*Software Defined Silicon*), илюстрирана на фиг. 1.7. Архитектурата осигурява пул от ресурси – апаратни нишки (процеси), канали, таймери, портове, генератори, които се интегрират в целевата паралелна система чрез езика *XC*. Така се осигурява единен унифициран поток при проектирането на хардуерните и софтуерните компоненти.

Определено може да се каже, че *XC*, като реализация на *CSP*, в голяма степен отговаря на изискването на Чарлз Хоар, автора на *CSP*: „*the highest goal of programming language design to enable good ideas to be elegantly expressed*".