Paralelní programování a procesor typu Transputer

Petr Přikryl

Klíčová slova: PARALELNÍ PROGRAMOVÁNÍ, SYNCHRONIZACE PROCESŮ, SDÍLENÉ PROSTŘEDKY, SEMAFOR, TRANSPUTER, OCCAM, KANÁL

1. Úvod

Cílem tohoto příspěvku je shrnout základy, ze kterých vychází paralelní programování, a představit Transputer jako perspektivní procesor pro výstavbu paralelních počítačů. Blíže určuje typ architektury takového paralelního počítače a seznamuje se základními rysy programovacích jazyků Occam a 3L Parallel C.

2. Vlastnosti paralelních programů

Paralelní programy se skládají z popisu určitého množství procesů, které mohou být prováděny paralelně.

Poznámka: při klasickém sekvenčním programování se nerozlišují pojmy **program a proces**. V oblasti paralelního programování je nutné chápat program jako popis algoritmu a proces jako vlastní děj zpracování dat. Rozdíl je pochopitelný uvědomíme-li si, že podle jednoho programu může běžet více procesů.

Na rozdíl od **sekvenčních algoritmů,** které vyžadují **úplné uspořádání** prováděných akcí, je u **paralelních** programů definováno jen **částečné uspořádání** prováděných akcí.

Příklad: mějme obraz, který lze rozdělit na 16 stejně velkých částí, a mějme 8 procesorů, které mohou paralelně zpracovávat jednotlivé části daného obrazu. Částečným uspořádáním prováděných akcí při paralelním zpracování se myslí skutečnost, že je nutné například **nejdříve** rozdělit obraz, teprve **potom** můžeme přistoupit k paralelnímu zpracování jednotlivých částí a **nakonec** sestavit celkový výsledek (transformovaný obraz). Tyto tři činnosti jako celky mají tedy pevně předepsané pořadí. Ale v rámci druhého kroku - zpracování částí obrazu - není předepsáno, která část obrazu se začne zpracovávat dříve a která později. I kdybychom měli dostatečný počet procesorů a kdyby se všechny části obrazu začaly zpracovávat ve

stejný čas, nemůžeme předem říct, u které části se zpracování dokončí nejdříve (doba zpracování může záviset na složitosti obrazu atp.).
Poznámka: Pokud je procesů více než fyzických procesorů, musí být pro vyhodnocování více procesů na jednom procesoru použita technika sdílení času (time slicing). Extrémním případem je situace známá z oblasti operačních systémů umožňujících víceuživatelský režim (multi-task) v případech, kdy máme k dispozici jediný procesor.

Považujeme-li zpracovanou část obrazu za výstup, může se při paralelním zpracování změnit časové pořadí výstupů proti vstupům. Při zpracování tedy vzniká **nedeterminismus.** Následkem nedeterminismu mohou ve špatně navrženém programu vznikat chyby závislé na shodě okolností. Ladění takového programu může být znemožněno skutečností, že programátor není schopen znovu navodit stejný chybový stav. V paralelním programu by se proto například neměla vyskytovat žádná část, jejíž chování by bylo závislé na čase.

Na průběh celého paralelního řešení problému se můžeme dívat jako na jeden proces, který jednou začal a vnitřně se rozštěpil na několik paralelních procesů. U smysluplných programů proto musí nutně dojít k situacím, kdy se některé procesy dostávají do vzájemného kontaktu nebo závislosti.
Interakce procesů se může vyskytnout za třech různých okolností:

a) když procesy žádají o přístup **ke sdíleným prostředkům** (těmi mohou být procesor, společná paměť, ...),
b) když procesy **potřebují časově sladit svou činnost,**
c) když si procesy chtějí mezi sebou **předat data.**

Ve všech třech případech musí dojít k **synchronizaci** činnosti procesů. Musí k ní dojít buď proto, aby se předešlo konfliktu (jako v případě (a)) nebo aby mohlo dojít ke kontaktu procesů (jako v případech (b) a (c)).

Při práci se sdílenými prostředky musí být zavedena určitá omezení, aby byla zajištěna "spravedlnost" při jejich přidělování a aby se bylo možné vyhnout stavu **uváznutí** (deadlock) nebo aby bylo možné se z tohoto stavu zotavit. **Uváznutí** je situace, kdy některý z procesů vlastní prostředky požadované jiným procesem a odmítá je uvolnit. Chovají-li se takto vzájemně minimálně 2 procesy, dostanou se do stavu nekonečného čekání.
Obecně existují dva důvody pro rozhodnutí využít paralelismus při řešení daného problému:

a) paralelismus je přímo obsažen v řešeném problému (představa paralelního řešení je z logického pohledu přijatelnější než řešení na sekvenčním počítači) - v takovém případě někdy mluvíme o **vrozeném paralelismu,**

b) je nutné (nebo možné) zrychlit řešení konkrétního problému tím, že jeho nezávislé části řešíme paralelně.

O případě (b) má význam uvažovat jen tehdy, máme-li k dispozici skutečně paralelní počítač, tj. počítač s několika procesory. Použití technik pro pseudoparalelní zpracování na jednom procesoru v případech, kdy existuje sekvenční algoritmus vede vždy k horším výsledkům. Pokud tomu tak není, určitě existuje lepší sekvenční algoritmus.

Příklady aplikací s vrozeným paralelismem:

- Systémy pracující v reálném čase-např. sběr a zpracování dat měřených paralelně pomocí velkého množství čidel
- Simulační systémy-paralelní procesy modelují činnost nějakého zařízení.

Příklad aplikace využívající paralelního algoritmu pro zvýšení výpočetního výkonu:

Násobení dvou matic - jednotlivým procesorům jsou rozeslány vždy řádek první a sloupec druhé matice. Každý procesor řeší skalární součin dvou vektorů a vrátí hodnotu prvku výsledné matice nadřazenému procesu.

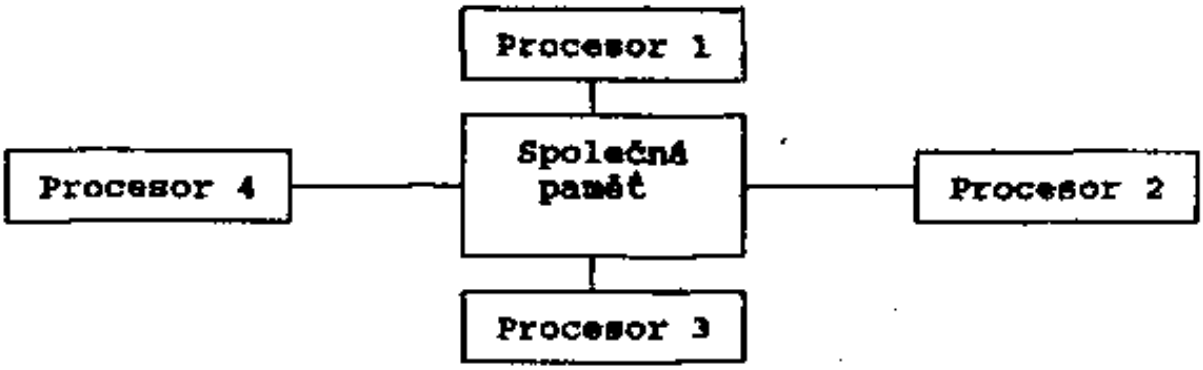
K poslednímu příkladu - hlediska konzervativního lidského uvažování může být člověku bližší sekvenční zpracování (prvek po prvku), ale i paralelní varianta je přijatelná, protože řešení je "průhledné" a na první pohled nedochází k žádným komplikacím. Nemusí tomu tak být u složitějších algoritmů. Mnohdy je nutné věnovat značné úsilí vlastnímu odhalení vnitřního paralelismu úlohy.

3. Architektury paralelních počítačů

Profesionální programátoři se obvykle neobejdou bez znalosti technického vybavení počítače, pro který je program určen. V oblasti paralelních počítačů jsou tyto znalosti ještě důležitější.

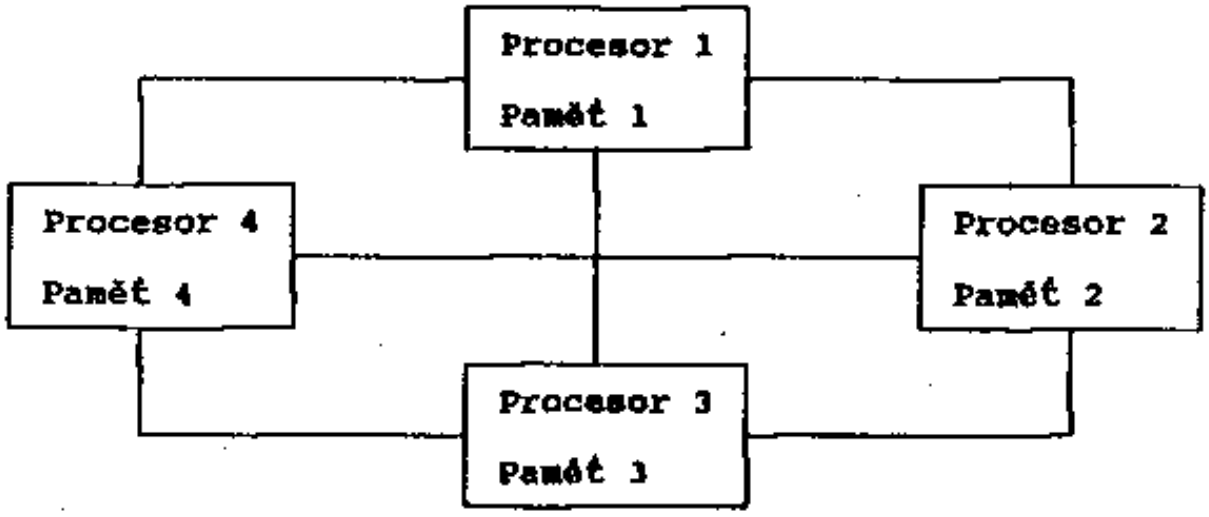
Pokud se mluví o víceúčelových paralelních počítačích, mluví se o kategorii označované jako MIMD (multiple instruction - multiple data). Jedná se o systém s více vzájemně propojenými procesory, z nichž každý je schopen zpracovávat jiná data podle jiného programu.

Jednu z významných architektur představuje tzv. **pevně vázaná** (tightly coupled) multiprocesorová architektura (viz obr. 1). Všechny procesory jsou stejného typu a pracují se sdílenou pamětí, ve které je uložen jak program, tak data. Výhodou takových systémů je jednoduchá výměna dat mezi různými procesy; nevýhody vyplývají ze skutečnosti, že výkon celého systému je omezen rychlostí společné paměti.



Obr.1 Pevně vázaná multiprocesorová architektura

Odlišným typem architektury je **volně vázaná multiprocesorová architektura** (loosely coupled multiprocessor architecture). Každý procesor má svou vlastní paměť a s ostatními procesory je spojen jen prostřednictvím komunikačních linek (viz obr.2).



Obr.2 Volně vázaná multiprocesorová architektura - úplné propojení procesorů

Jiná varianta volně vázané multiprocesorové architektury využívá vzájemného propojení procesorů přes společnou přepínací síť.

Další typ architektury obdržíme při kombinaci pevně a volně vázané architektury (dále také architektury s pevně a volně vázanými procesory). Vznikne **hybridní architektura.** Každý procesor má svou lokální paměť a navíc existuje paměť společná pro

všechny procesory. Přes ni všechny procesory komunikují a v ní se udržují sdílené datové struktury.

Použití každé z uvedených architektur může ovlivnit způsob psaní paralelních programů. Pro architekturu s pevně vázanými procesory můžeme synchronizaci procesů zajistit při výlučném přístupu do společné paměti, u architektury s volně vázanými procesory budou převažovat problémy s komunikací procesorů po linkách.

Z hlediska programátora lze tyto rysy architektur odstínit snad jen v některých triviálních případech. Znalost typu architektury a zvláště konkrétní konfigurace či topologie už při psaní programu může významným způsobem ovlivnit jeho vlastnosti. Další viz [1].

4. Transputer

Transputer je perspektivní součástka navržená britskou firmou INMOS. Tak například na jednom čipu (typ T800) vyrobeném technologií VLSI je integrován velmi výkonný 32-bitový procesor typu RISC, rychlá statická paměť RAM (4kB), jednotka pro výpočty v pohyblivé řádové čárce a jednotka pro komunikaci po čtyřech obousměrných sériových linkách. K čipu lze připojit další paměť (typicky 1MB). Z hlediska programování na nižší úrovni se jedná o samostatný počítač spojený s "okolním světem" pouze pomocí čtyř linek. Každá linka slouží pro spojení pouze s jedním sousedním procesorem nebo periferním zařízením. Typickým místem použití Transputeru tedy jsou multiprocesorové systémy s volně vázanými procesory bez propojovací sítě (viz obr.2). Propojením procesorů "od bodu k bodu" odpadá řada problémů typu sdílení sběrnice, sdílení paměti, řešení přepínací sítě atd. což jsou u paralelních systémů nejsložitější problémy. Další viz [7].

Systémy s Transputery se komerčně dodávají v podobě desky připojitelné k osobnímu počítači. Nejmenší deska obsahuje jeden Transputer, často se používá deska se čtyřmi Transputery.

Souběžně s návrhem Transputeru byl navržen i programovací jazyk Occam. Skutečnost, že byl procesor a jazyk navrhován současně vedla k tomu, že prostředky programovacího jazyka odpovídají technickým prostředkům procesoru a naopak. Transputer proto bývá označován za tzv. Occam-Engine i když je nutné zdrojový text v Occam-u nejdříve zkompilovat.

5. Occam (INMOS Limited TM)

Occam je **vyšší programovací jazyk** navržený pro zápis paralelních algoritmů a pro jejich implementaci na síti procesorů. Occam je postaven na pevných teoretických základech a byl proto rychle přijat nejen jako jazyk pro programování, ale také jako mocný prostředek pro specifikaci problémů a pro popis chování paralelních systémů.

Occam představuje ideální metodologický úvod do paralelního programování. Program psaný v Occam-u poskytuje stupeň bezpečnosti neznámý v konvenčních programovacích jazycích jako jsou C, Pascal, Fortran.

Jazyk Occam je pojmenován po filozofu 14. století Williamu z Occamu. V duchu jeho filozofie byl jazyk založen na principech minimalizace jazykových mechanismů.

Occam-ovský model klade důraz na **paralelismus a komunikaci.** Stavební jednotkou z hlediska zpracování je proces, základní a jedinou jednotkou pro komunikaci je kanál. Occam umožňuje hierarchickou výstavbu systému. Množinu procesů spojených kanály můžeme označit jako jediný proces vyšší úrovně.

Kanál je jednosměrná datová cesta spojující právě dva procesy. Ze strany jednoho procesu je kanál používán vždy jako výstupní, ze strany druhého procesu je vždy jako vstupní. Komunikace po kanále probíhá bez vyrovnávací paměti. To znamená, že proces vysílající data musí pozastavit svou činnost až do doby, kdy je proces na druhé straně kanálu schopen data přijmout. Podobně, dojde-li k operaci čtení z kanálu dříve než jsou data na straně vysílajícího procesu připravena, je čtení pozastaveno (tím i celý proces) až do doby, kdy druhý proces data vyšle. Takovým způsobem je zajištěna synchronizace procesů typu (b) a (c) (viz výše). Žádné dva procesy nesdílí společnou paměť. Odpadá proto nutnost synchronizace typu (a).

V Occam-u jsou definovány následující primitivní procesy:

- přiřazení a paralelní přiřazení (např. **a,b,c := 1,2,3**),
- procesy komunikace:
  vstup (příjem hodnoty z kanálu - např. **kanal ? prom**)
  výstup (odeslání hodnoty - např. **kanal ! 3**),
- SKIP - proces začne, nic nedělá, skončí,
- STOP - proces začne, nic nedělá a nikdy neskončí.

Procesy se spojují do nově definovaného, hierarchicky vyššího procesu pomocí následujících konstrukcí:

- **SEQ** - určuje posloupnost procesů, které se mají provést sekvenčně.
- **IF** - konstrukce pro podmíněný proces - posloupnost dvojic "booleovský výraz", "proces". Podmínky se vyhodnocují v pořadí v jakém byly napsány; první splněná

podmínka vede ke spuštění příslušného procesu; není-li splněna ani jedna podmínka, chová se konstrukce jako primitivní proces STOP.

- **CASE** - konstrukce výběru - na základě hodnoty výrazu-selektoru je spuštěn jemu odpovídající proces; nesouhlasí-li hodnota výrazu s žádnou hodnotou ve funkci návěští, chová se konstrukce jako primitivní proces STOP.
- **WHILE** - konstruktor cyklu.
- **PAR** - určuje procesy, které se mají provádět současně.
- **ALT** - konstrukce pro spuštění procesu podmíněného splněním strážní podmínky. Zápis obsahuje posloupnost dvojic "strážní podmínka", "proces". Strážní podmínkou je primitivní proces vstupu z kanálu případně podmíněný booleovským výrazem nebo booleovská podmínka spojená s primitivním procesem SKIP. Splněním strážní podmínky se rozumí situace, kdy lze realizovat vstup s kanálu (data jsou dostupná) a je současně splněna booleovská podmínka (je-li uvedena),případně je-li splněna booleovská podmínka spojená s procesem SKIP.

Proměnné musí být vždy deklarovány před použitím a jsou dostupné pouze z konstrukce u které se jejich deklarace objevila. Jednoduché proměnné mohou být booleovské, celočíselné a reálné o různé délce. Jednoduché datové typy mohou být použity pro definici typu pole, záznam a pro definici proměnné typu kanál (CHAN) - resp. k definici protokolu kanálu. Dalším speciálním datovým typem je typ časovač (TIMER). K deklarované proměnné tohoto typu se chováme jako ke kanálu, na jehož druhém konci fiktivní proces v čase čtení vyšle aktuální systémový čas.

Occam umožňuje definovat procedury a funkce. Dalším prostředkem pro psaní programů je mechanismus tzv. zkratek, který připomíná makro-jazyk. Pomocí zkratek lze například vyjádřit hodnoty konstant, přejmenování proměnných, pojmenování částí polí. Charakter použití maker má také tzv. replikace. Replikovat lze konstrukce SEQ, IF, PAR a ALT. Replikovaná konstrukce připomíná zápis cyklu 'for' známý z obecných programovacích jazyků.

Program psaný v Occamu se chová až na rychlost stejně je-li spuštěn na jednoprocesorovém systému nebo na systému s více Transputery. Prostředky jazyka lze určit přiřazení jednotlivých procesů fyzickým procesorům a logických kanálů fyzickým kanálům. K tomu je nutné znát topologii procesorové sítě. Toto přiřazení se provádí obvykle až po dokonalém odladění verze pro jeden procesor. Další informace lze nalézt v [2].

6. 3L Parallel C (INMOS Limited TM)

Programovací jazyk 3L Parallel C je dodáván jako programový balík ve verzích MS-DOS a UNIX. Je odvozen od již klasické definice programovacího jazyka C (Kernighan, Richie). Jazyk není z hlediska paralelního programování tak bezpečný jako Occam. V Parallel C se rozlišují 2 typy procesů. Prvním typem je tzv. TASK, což je samostatně kompilovatelná programová jednotka. S okolím je tento typ procesu spojen pomocí vstupních a výstupních portů připomínajících co do funkce Occam-ovské kanály. Druhým typem procesu je tzv. THREAD. Jedná se o proces, který byl dynamicky vytvořen uvnitř procesu typu task. Každý proces typu thread má vlastní zásobník, ale sdílí svůj kód, statická data a prostor označovaný jako heap. Výlučný přístup ke sdíleným prostředkům je zajišťován mechanismem známým pod pojmem semafor.

Paralelní procesy typu task a thread, funkce pro práci se semaforem - to vše představuje mocné rozšíření jazyka C, ale nesprávné použití těchto prostředků vede k těžko odhalitelným chybám. Na rozdíl od paralelních konstrukcí v Occam-u nemůže být správnost jejich použití kontrolována překladačem.

Samotný překladač Parallel C neumožňuje vyjádřit fyzické přiřazení procesů a kanálů procesorům a fyzickým linkám. Toto přiřazení je nutné definovat pomocí tzv. konfiguračního souboru a provede se až v době sestavování kódu. Podrobnosti viz [3].

7. Závěr

Transputer se díky své promyšlené koncepci a velmi jednoduchému rozhraní (4 linky) stal prvkem pro výstavbu výkonných a hlavně dostupných paralelních počítačů. Kromě vývojových desek se objevují první vážné aplikace např. pro zpracování obrazu v reálném čase a v jiných oblastech náročných na výpočetní výkon. To je technická stránka vlivu existence Transputeru. Při pohledu na tyto skutečnosti z druhé strany - lze očekávat, že dostupnost takových systémů povede k rozmachu paralelního programování, k prohloubení jeho teoretických základů. Paralelní programování přestává být doménou laboratoří bohatých firem a organizací. Bylo by vhodné připravit se na to, že brzy zasáhne i nás a nenechat si v této nové oblasti ujet vlak.

Literatura

[1] Bustard, Elder, Welsh: Concurrent Program Structures, 1988, Pretience Hall, C.A.R. Hoare Series Editor, ISBN 0-13-167080

[2] INMOS Limited: Occam 2 - Reference Manual 1988, Pretience Hall, C.A.R. Hoare Series Editor, ISBN 0-13-629312-3

[3] INMOS Limited: 3L Parallel C - User Guide 1989, firemní publikace 72 TDS 179 00

[4] INMOS Limited: 3L Parallel C - Delivery Manual 1989, firemní publikace 72 TDS 182 00

[5] Distributed Software Limited ///:
The Helios Parallel Programming Tutorial, 1990, Helios Technical Guides

[6] Distributed Software Limited ///:
The CDL Guide, 1990, Helios Technical Guides

[7] INMOS Limited: Transputer Reference Manual

Autor: Ing. Petr P ř i k r y l,
Katedra informatiky a výp. techniky
FE VUT, Božetěchova 2, 612 66 Brno 12,
tel. 05-746 111/kl. 31