

# Implementace algoritmu Parallel Splitting

(projekt do předmětu PDA)

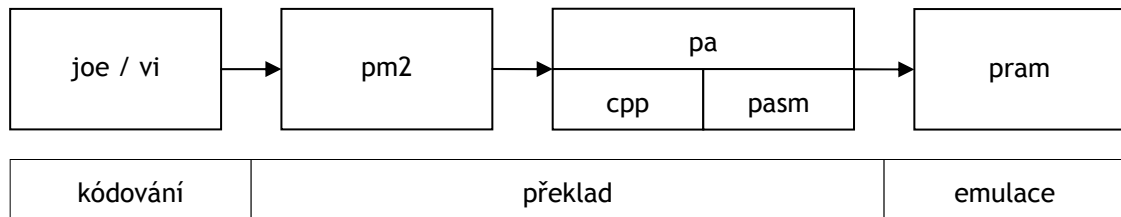
## Obsah

Obsah .....	1
Zadání projektu .....	2
Cíl projektu .....	2
Časový harmonogram řešení projektů .....	2
Odevzdání projektů: .....	2
Řešení projektu.....	3
Použitý systém .....	3
Implementace algoritmu .....	3
Experimentování .....	3
Závěr.....	4
Zdrojový kód programu .....	5
Použitá literatura .....	7

# Zadání projektu

## Cíl projektu

Cílem projektu je implementovat dva algoritmy pro paralelní zpracování dat. Algoritmy budou implementovány v jazyce *pm2* a spuštěny emulátorem *pram*. Cesta od napsání kódu k emulaci je ukázána na následujícím obrázku:



Kód programu lze přeložit do jazyka *PEL* (jazyk obdobný jazyku symbolických instrukcí) překladačem *pm2*. Dále by měl proběhnout překlad z jazyka *PEL* do binárního formátu. Dávka *pa* provádí tento překlad ve dvou krocích, a to nejprve spuštěním preprocesoru *cpp* na zadaný soubor a jeho následný překlad assemblerem *pasm*. Pokud překlad proběhne bez chyby, lze výsledný kód spustit emulátorem *pram*.

## Časový harmonogram řešení projektů

### 1. fáze, do 18.11.2004

V první fázi řešení projektu se proto seznámte s jazyky *pm2*, případně *PEL* a instalujte (přeložte) potřebné programy. Tato fáze nemá výsledek ve formě zprávy či aplikace, ale má sloužit jako příprava na další úkoly. Důkladně proto ověřte chování překladačů a emulátoru v prostředí, které se rozhodnete používat.

### 2. fáze, do 9.12.2004

Implementace algoritmu Merge-splitting Sort.

V prostředí jazyka *PM2* implementujte algoritmus Merge-splitting Sort a přeložte jej a následně ověřte jeho chování pomocí emulátoru *Pram*. Velikost vstupních dat i princip rozdělení mezi procesory si řešitel zvolí sám. Zvoleny postup ale musí být popsán v dokumentaci.

### 3. fáze, do 23.12.2004

Implementace algoritmu Parallel Splitting.

Pro vstupní posloupnost a zadanou hodnotu proveďte rozdělení algoritmem Parallel Splitting v prostředí *PM2*. Ověřte funkčnost algoritmu a proveďte srovnání časové složitosti implementace s výsledky analytického rozboru časové složitosti tohoto algoritmu.

## Odevzdání projektů:

V papírové podobě průběžně na přednáškách (pro 2. a 3. fázi v den termínu, tj. 9.12.2004 a 23.12.2004). Obsahem práce bude komentovaný výpis algoritmu v jazyce *PRAM*, stručný rozbor algoritmu a jeho analýza a dále rozbor provedených experimentů.

**POZOR!** Druhý projekt odevzdávejte prosím v kanceláři B-118. Elektronické odevzdání proběhne přes informační systém. Bude odevzdán archiv (jmeno.arj / jmeno.zip / jmeno.rar), který bude obsahovat komentovaný zdrojový kód *PM2*, dokumentaci odpovídající papírové formě a případně i vstupní data, pro které byl algoritmus testován.

14.12.2004, František Zbořil ml.

# Řešení projektu

## Použitý systém

Programové vybavení zodpovídalo tomu, které se použilo v prvním úkolu, detaily tudíž v [3].

## Implementace algoritmu

Po seznámení se s algoritmem, převážně ze slajdů [2], jsem navrhnul první variantu která by šla implementovat pomocí *pm2*.

Jednalo se o rozdělení vstupního souboru dat na části podle procesoru a pak paralelní výpočet posloupností L, E a G. Tato varianta by ale vyžadovala hodně paměti jelikož je třeba alokovat nejméně tolik paměti pro každý výstupní vektor jak je dlouhé vstupní pole a dále taky lokální proměnné udržující délku zvláště pro každý procesor, tj. vzorcem by to bylo  $3N + 3P$ .

Začal jsem teda s optimalizacemi - první co jde odstranit je vektor E a nahradit ho jenom jeho velikostí a tím se paměťová náročnost zmenší na  $2N + 3P$ . Po dalším uvažování jsem přišel na fakt, že vlastně není potřeba nikam ukládat ty zpracované vektory, protože potřebujeme jenom velikost každé kategorie pro každý procesor, tj. jenom  $3P$  paměti. Po tom co spočteme délky L, E a G vektorů je musíme agregovat abychom dostali cílové indexy.

Tato agregace je vlastně suma prefixů, ale pro jednoduchost implementace jsem provedl výpočet jenom na 1 procesoru, tj. sériově. Zbylé procesory nic nedělají. Tuto cestu jsem volil taky z toho důvodu, že pro testování jsem použil proměnlivý počet procesorů (1 až 32) a sestavovat výpočetní strom sumy prefixů za běhu by bylo náročné. U konkrétního případu, kde se nachází přesně  $2^x$  procesorů není problém sestavit strom který zaručí časovou náročnost kroku  $\log N$ .

Po výpočtu indexů se provádí znova paralelní operace, tentokrát ale jde o operaci přepisu vstupního pole na dvě výstupní (L a G). Pole E je nahrazeno jeho velikostí. Data po výpočtu se z programu dostanou jednoduchým výpisem ve tvaru:

```
délka L
délka E
délka G
data L
data E
data G
```

## Experimentování

Výše zmíněný výpis se dá zpětně načíst do testovacího programu který prováděl hromadné testování programu na rozdělování pole. Po ověření základní správnosti výsledku dle vzorců:

$$\max(L) < PoS < \min(G)$$

$$\max(E) = PoS = \min(E)$$

se výsledek bere za platný a započte se (L,E a G jsou vektory dat podle kategorií, PoS je bod rozdělení, point of split).

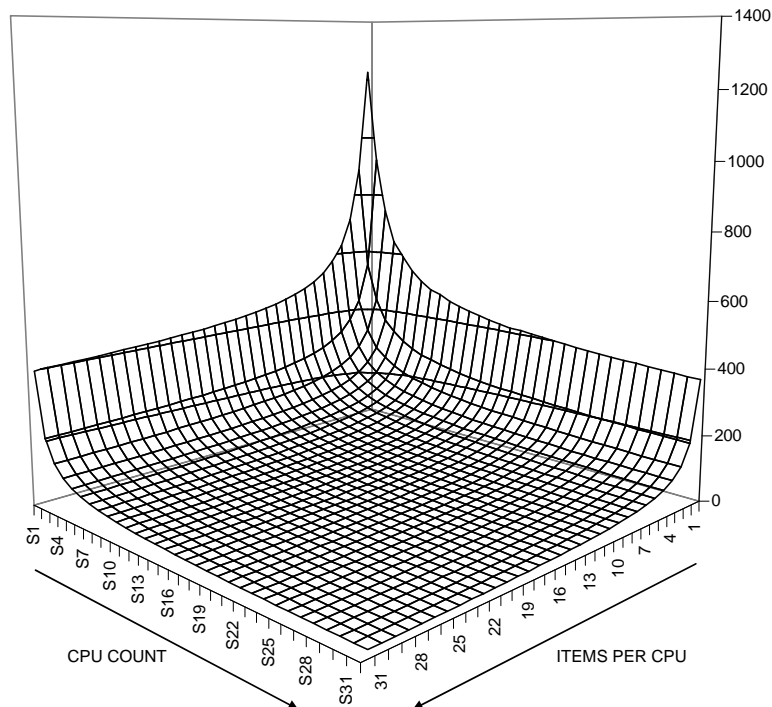
Detaily experimentu:

- 1 až 32 procesorů
- 1 až 32 prvků pole na procesor
- výsledek je průměrem 5-ti pokusů
- pro větší věrohodnost se ze stejnými daty provedl program bez výpočtu a takto zjištěný čas odečetl od programu s výpočtem

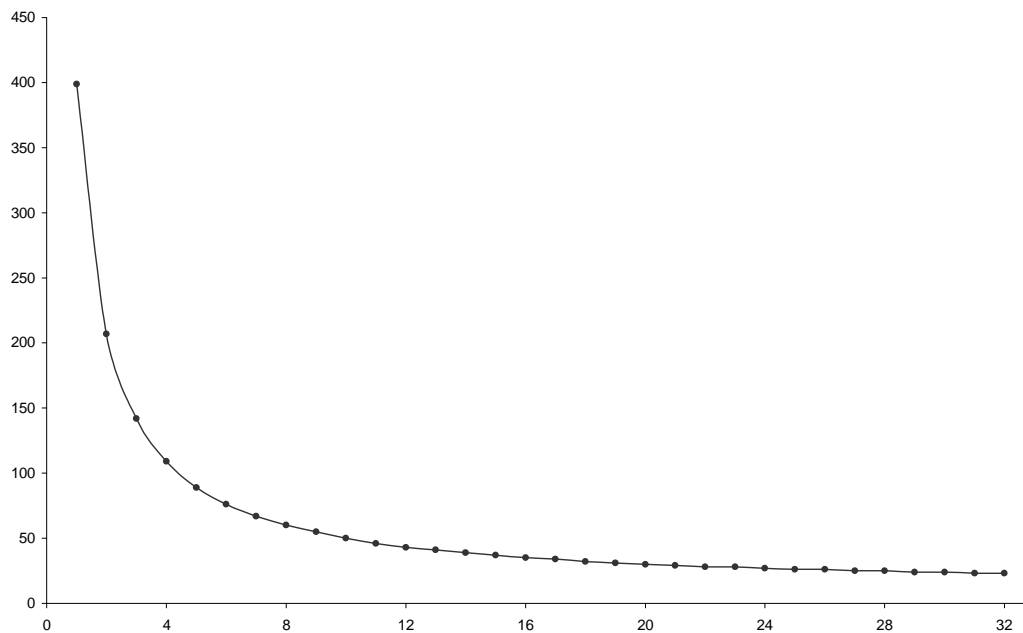
Výpočet trval 34 minut 57 sekund když byl puštěn na procesoru Intel Celeron Tualatin taktovaném na 1568 MHz (jádro PIII, 256 kB L2 cache).

## Závěr

Data zjištěná z hromadného testu programu pomocí náhodně vygenerovaných vstupních dat se dají nejlíp zobrazit pomocí 3D grafu (výška značí počet taktů na prvek vstupních dat, tj. jde o poměrnou časovou náročnost):



Graf je velice hladký, což naznačuje že se algoritmus chová hodně deterministicky. Na průběhu grafu ze strany kde se zvyšuje počet procesorů je vidět funkce  $1/X$ :



Při porovnání s analytickým řešením náročnosti algoritmu ( $N/P$ ) se dá odvodit, že náročnost v praxi souhlasí s náročností teoretickou. Podrobněji: Graf má výšku  $Z$ , která odpovídá  $T/N$ , takže tady se ruší  $N$  v čitateli, a  $X$  ve jmenovateli zlomku je právě to  $P$ .

## Zdrojový kód programu

```
program parallelSplitting;

const      p      = 4;                // processors
const      nPP   = 3;                // n per processor
const      n      = p*nPP;           // total number of elements
sharedvar  d      : array[0..n-1] of word; // source data
           PoS    : word;            // point of split
           L      : array[0..n-1] of word; // destination data
           G      : array[0..n-1] of word; // destination data
           Ln     : word;            // number of elements
           En     : word;
           Gn     : word;
           La     : array[0..p-1] of word; // count per processor
           Ea     : array[0..p-1] of word;
           Ga     : array[0..p-1] of word;
           Lz     : array[0..p-1] of word; // index per processor
           Ez     : array[0..p-1] of word;
           Gz     : array[0..p-1] of word;

procedure init;
begin
    // CPU count
    P := p;
    // pre-generated input
    d[ 0] := 12;
    d[ 1] := 9;
    d[ 2] := 10;
    d[ 3] := 11;
    d[ 4] := 7;
    d[ 5] := 4;
    d[ 6] := 3;
    d[ 7] := 6;
    d[ 8] := 2;
    d[ 9] := 1;
    d[10] := 8;
    d[11] := 5;
    // point of split
    PoS := 5;
    // result initialization
    Ln := 0;
    En := 0;
    Gn := 0;
end init;

procedure finish;
var i : word;
begin
    // output format: countL,countE,countG,dataL*,dataE*,dataG*
    writeln( Ln );
    writeln( En );
    writeln( Gn );
    // LESS
    if Ln > 0 then
        for i := 0 to Ln-1 do
            writeln( L[i] );
        end;
    end;
    // EQUAL
    if En > 0 then
        for i := 1 to En do
            writeln( PoS );
        end;
    end;
    // GREATER
    if Gn > 0 then
        for i := 0 to Gn-1 do
            writeln( G[i] );
        end;
    end;
end finish;
```

```

procedure countElements( cpu,b,e,split : word );
var i : word;
begin
    for i := b to e do
        if d[i] = split then
            Ea[cpu] := Ea[cpu] + 1;
        else
            if d[i] < split then
                La[cpu] := La[cpu] + 1;
            else
                Ga[cpu] := Ga[cpu] + 1;
            end;
        end;
    end;
end countElements;

procedure flushElements( b,e,split,Ldi,Gdi : word );
var i : word;
begin
    for i := b to e do
        if d[i] <> split then
            if d[i] < split then
                L[Ldi] := d[i];
                Ldi := Ldi + 1;
            else
                G[Gdi] := d[i];
                Gdi := Gdi + 1;
            end;
        end;
    end;
end flushElements;

var i,j : word;
begin

    // initialization of per-processor variables [PAR]
    par i := 0 to p-1 do
        La[i] := 0;
        Lz[i] := 0;
    end;
    synchronize;

    // count the elements of each class (L,E,G) [PAR]
    par i := 0 to p-1 do
        countElements(i,i*nPP,(i+1)*nPP-1,PoS);
    end;
    synchronize;

    // calculate the indexes a->z [SER]
    par j := 0 to 0 do
        if p > 1 then
            for i := 0 to p-2 do
                Lz[i+1] := Lz[i] + La[i];
                Ez[i+1] := Ez[i] + Ea[i];
                Gz[i+1] := Gz[i] + Ga[i];
            end;
        end;
        i := p-1;
        Ln := Lz[i] + La[i];
        En := Ez[i] + Ea[i];
        Gn := Gz[i] + Ga[i];
    end;
    synchronize;

    // flush the result [PAR]
    par i := 0 to p-1 do
        flushElements(i*nPP,(i+1)*nPP-1,PoS,Lz[i],Gz[i]);
    end;
    synchronize;

    // against the strange error when pram prints multiple clock lines
    synchronize;

@CLOCK

end parallelSplitting.

```

## Použitá literatura

- [1] **PDA - Zadání projektu**  
[http://www.fit.vutbr.cz/~zborilf/study/PDA/pda\\_projekt04.htm](http://www.fit.vutbr.cz/~zborilf/study/PDA/pda_projekt04.htm)
  
- [2] **Vektorové a maticové algoritmy - slajdy**  
<https://www.fit.vutbr.cz/study/courses/PDA/private/www/h005.pdf>
  
- [3] **Rozsnyó, D.: Implementace algoritmu Merge-splitting Sort (Projekt do předmětu PDA)**  
<http://rozsnyo.com/diary/2004/12/09/PDA1.pdf>