SINHRONIZIRANA PODATKOVNO PRETOKOVNA RAČUNALNIŠKA ARHITEKTURA II

INFORMATICA 3/92

Jurij Šilc
Laboratorij za računalniške arhitekture
Inštitut Jožef Stefan, Ljubljana
Ludvik Gyergyek
Fakulteta za elektrotehniko in računalništvo,
Ljubljana

Keywords: parallel processing, dataflow computing, scheduling, allocator, performance evaluation, parallel computer architecture

Delo obravnava problem časovne optimizacije asinhronega procesiranja na omejenem številu procesorjev. Predlagamo izvirno rešitev, ki temelji na uvedbi nekaterih mehanizmov sinhronizacije v asinhrono računanje. Graf pretoka podatkov, ki opisuje asinhrono procesiranje, opremimo s časovno optimalno sprožitveno funkcijo, ki služi tako pri vlaganju grafa v računalnik, kakor tudi pri njegovem časovno optimalnejšem izvrševanju. V ta namen smo razvili hevristična algoritma `pOptSinh` in `TOptSinh` za konstrukcijo optimalnih sprožitvenih funkcij, ki po pesimistični oceni vračata optimalno rešitev v 80% primerov. Nadalje predlagamo algoritma za dodeljevanje `MinGlDol` in `MinGlGor`, ki temeljita na optimizaciji medprocesorskih komunikacij. V primerjavi z znanimi razvrščevalnimi algoritmi dobimo s predlaganima algoritmoma boljše rezultate, kar potrjujejo analizirani primeri algoritmov za izračun hitre Fourierjeve transformacije, dinamične analize scene in LU razcepa matrike. Končno podajamo tudi zasnovo hibridne vzporedne arhitekture računalnika, ki podpira predlagano preoblikovanje asinhronega računanja.

SYNCHRONOUS DATAFLOW COMPUTER ARCHITECTURE - We discuss the problem of time optimization of asynchronous processing on a limited number of processors. We present an original solution to the problem based on introduction of synchronization mechanisms into asynchronous processing. The dataflow graph describing asynchronous processing is associated with the corresponding time-optimal firing function. This function is used both for loading a dataflow graph into the computer and for time-optimal graph execution. In order to do this, we have developed two heuristic algorithms, `pOptSinh` and `TOptSinh`, which are used for optimal firing function construction. According to conservative estimates, these algorithms return optimal functions with 80% probability. Furthermore, we propose two scheduling algorithms, `MinGlDol` and `MinGlGor`, which are based on interprocessor communication minimization. These two algorithms give better results compared to some other well known scheduling algorithms. This fact is illustrated in Fast Fourier Transformation, Dynamic Scene Analysis, and LU matrix decomposition algorithms. Finally, we present a design of hybrid parallel computer architecture capable of supporting modified asynchronous computing.

1. Asinhrono računanje

V nadaljevanju bomo opisali način *asinhronega* računanja, ki ga bomo opisali s pomočjo *GPP*. Formalizirali bomo problema minimizacije časa oziroma procesorjev. Uvedli bomo pojem sprožitvene funkcije, ki nam bo v nadaljevanju omogočil vpeljati nekatere mehanizme *sinhronizacije* v asinhrono računanje.

1.1 Graf pretoka podatkov

Graf G, ki ga sestavljata množici *točk* $\mathcal{V}$ in *povezav* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, označimo z $G(\mathcal{V},\mathcal{E})$. Število točk in povezav označimo z $n = |\mathcal{V}|$ oziroma $m = |\mathcal{E}|$. Kadar je $(u,v) \in \mathcal{E}$, pravimo, da je točka v *neposredni naslednik* točke u, in to označimo z $u \longmapsto v$. Kadar obstaja vsaj ena usmerjena pot iz točke u do točke v, pa pravimo, da je v *naslednik* točke u, kar označimo z $u \overset{+}{\longmapsto} v$. Relacija $\overset{+}{\longmapsto}$ je *tranzitivna ovojnica* relacije $\longmapsto$. Graf G je *acikličen*, če ne vsebuje točke v z lastnostjo $v \stackrel{+}{\longmapsto} v$. Če je G acikličen, je $\longmapsto$ *relacija stroge delne urejenosti* v $\mathcal{V}$, saj je irefleksivna, asimetrična in tranzitivna. *Vhodnja stopnja* $d^+(v)$ točke v je število povezav, ki se stekajo v v, *izhodnja stopnja* $d^-(v)$ točke v pa je število povezav, ki točko v zapuščajo. Pravimo, da je G *enostaven*, če obstaja med dvema točkama največ ena povezava v isti smeri.

Naj bo $G(\mathcal{V}, \mathcal{E})$ acikličen. Tedaj zapišemo $\mathcal{V}$ kot:

$$\mathcal{V} = \mathcal{V}_Z \cup \mathcal{V}_N \cup \mathcal{V}_K.$$

$\mathcal{V}_Z$, $\mathcal{V}_N$ in $\mathcal{V}_K$ se imenujejo množica *začetnih*, *notranjih* oziroma *končnih* točk. Velja naslednje:

$$\begin{aligned} \forall v \in \mathcal{V}_Z &: \quad d^+(v) = 0 \wedge d^-(v) > 0, \\ \forall v \in \mathcal{V}_N &: \quad d^+(v) > 0 \wedge d^-(v) > 0, \\ \forall v \in \mathcal{V}_K &: \quad d^+(v) > 0 \wedge d^-(v) = 0. \end{aligned}$$

Če sta $\mathcal{V}_Z$ in $\mathcal{V}_K$ končni, neprazni in nevezani množici, potem v G ni nobene izolirane točke.

V nadaljevanju bomo obravnavali le grafe pretoka podatkov GPP, ki so v skladu s sledečo definicijo:

DEFINICIJA: Par $(G(\mathcal{V}, \mathcal{E}), t)$ je *graf pretoka podatkov*, če velja:
(a) G je usmerjen, enostaven in acikličen;
(b) $\mathcal{V}_Z \cap \mathcal{V}_K = \emptyset$;
(c) $t : \mathcal{V} \longrightarrow \mathbb{N}$. □

Če je $v \in \mathcal{V}$, pomeni $t(v)$ *čas izvrševanja* točki v pridružene operacije[1].

1.2 Najkrajši čas izvrševanja

Naj bodo dani $GPP = (G, t)$ ter točki u in v. Če velja $u \stackrel{+}{\longmapsto} v$, obstaja iz točke u do v vsaj ena *pot* $\wp = w_1, w_2, \ldots, w_k$, kjer je $w_1 = u$ ter $w_k = v$. Tedaj definiramo *dolžino poti* $\wp$ kot $\ell(\wp) = \sum_{i=1}^{k} t(w_i)$. Dolžino *najdaljše* poti iz točke u v točko v pa označimo z $\ell(u, v)$. Pot iz množice točk $\mathcal{U}$ v množico točk $\mathcal{W}$ je vsaka pot, ki se prične v eni od točk množice $\mathcal{U}$ in se konča v eni od točk množice $\mathcal{W}$. Dolžino najdaljše poti med množicama $\mathcal{U}$ in $\mathcal{W}$ pa označimo z $\ell(\mathcal{U}, \mathcal{W})$. Če vsebuje kaka od množic $\mathcal{U}$ ali $\mathcal{W}$ en sam element, pišemo namesto množice kar sam element; na primer, če je $\mathcal{U} = \{u\}$, pišemo kar $\ell(u, \mathcal{W})$.

[1] Operacij posebej ne navajamo, ker za samo analizo niso pomembne.

Seveda je $\ell(\mathcal{V}_Z, \mathcal{V}_K)$ *najkrajši* možni čas, v katerem se še more izvršiti GPP. Označimo ga s T_∞, ker se GPP izvrši v najkrajšem možnem času le ob zadosti velikem (praktično "neskončnem") številu procesorjev. Če je na voljo le en procesor (zaporedno izvrševanje), pa označimo najkrajši možni čas za izvršitev GPP s T_1. Seveda velja $T_1 = \sum_{v \in \mathcal{V}} t(v)$.

1.3 Sprožitvena funkcija

Naj bodo dani $GPP = (G, t)$, naravno število $T \geq T_\infty$ ter funkcija $s : \mathcal{V} \longrightarrow \{0, 1, \ldots T\}$. Če je $v \in \mathcal{V}$, pišemo $f(v) = s(v) + t(v)$.

DEFINICIJA: Funkcijo s imenujemo *sprožitvena* funkcija, če velja:
(a) $f(v) \leq T, \quad \forall v \in \mathcal{V}$ in
(b) $u \longmapsto v \Longrightarrow f(u) \leq s(v), \quad \forall u, v \in \mathcal{V}$. □

Sprožitvena funkcija s priredi vsaki točki v trenutek *sprožitve* $s(v)$, v katerem točka v zajame vhodne podatke in prične izvrševanje pridružene operacije. Posredno je s tem natanko določen tudi trenutek *izstrelitve* $f(v)$, v katerem točka v konča svoje izvrševanje in pošlje podatke vsem neposrednim naslednikom. Zgornja definicija zagotavlja, da vsaka točka izstreli svoje rezultate pred trenutkom T; poleg tega pa proženje ter izstreljevanje spoštujeta relacijo $\longmapsto$. Za dani GPP v splošnem obstaja več različnih sprožitvenih funkcij. Množico vseh sprožitvenih funkcij danega GPP označimo z $\mathcal{S}(T)$. Vsaka $s \in \mathcal{S}(T)$ porodi pravilo, po katerem poteka računanje GPP. Kadar bomo želeli to pravilo posebej poudariti, bomo k oznaki s dodali ustrezni indeks. Sprožitveno funkcijo s imenujemo *požrešna*, če velja

$$s(v) = \ell(\mathcal{V}_Z, v) - t(v), \text{ za vsak } v \in \mathcal{V},$$

oziroma *lena*, če velja

$$s(v) = T - \ell(v, \mathcal{V}_K), \text{ za vsak } v \in \mathcal{V}.$$

Požrešno[2] sprožitveno funkcijo bomo označili s s_e, leno[3] pa s s_l. Prva opisuje podatkovno vodeno računanje, druga pa računanje vodeno z zahtevo.

[2] Požrešno iz angl. "eager".
[3] Leno iz angl. "lazy".

1.4 Kritične točke

Naj bo dan $GPP = (G, t)$. Poti z dolžino $\ell(\mathcal{V}_Z, \mathcal{V}_K)$ so *najdaljše* poti v GPP.

DEFINICIJA: Točko v imenujemo *kritična*, kadar se nahaja na kaki od najdaljših poti v GPP. Če je $0 \leq \tau \leq \ell(\mathcal{V}_Z, \mathcal{V}_K)$, imenujemo točko v *kritična na globini* τ, kadar je kritična in velja $\ell(\mathcal{V}_Z, v) - t(v) = \tau$. □

Množico kritičnih točk označimo s $\mathcal{K}$, množico kritičnih na globini τ pa s $\mathcal{K}(\tau)$. Kritičnost je torej značilnost GPP samega, saj je moč enostavno pokazati, da je kritična natanko tista točka $v \in \mathcal{V}$, za katero velja

$$\ell(\mathcal{V}_Z, v) - t(v) = \ell(\mathcal{V}_Z, \mathcal{V}_K) - \ell(v, \mathcal{V}_K).$$

V posebnem primeru smemo kritičnost točk opredeliti tudi z vidika sprožitvenih funkcij. Naj bo dan čas izvrševanja T. Opazimo, da je levi del zgornje enačbe enak $s_e(v)$. Desni del pa je enak $s_l(v)$, vendar le, če je $T = T_\infty$! V tem primeru je kritični točki možno prirediti le en in en sam trenutek za njeno sprožitev, neodvisno od izbrane sprožitvene funkcije $s \in \mathcal{S}(T_\infty)$. Značilnost točk $v \in \mathcal{K}(\tau)$ pa je, da se sprožijo natanko v trenutku τ, zato jih včasih imenujemo tudi *kritične v trenutku* τ.

1.5 Zahteve po procesorjih

Naj bodo dani $GPP = (G, t)$, naravno število $T \geq T_\infty$ ter sprožitvena funkcija $s \in \mathcal{S}(T)$. Kadar je $\tau \in \{0, 1, \ldots, T\}$, definiramo

$$p_{T,s}(\tau) = \Big|\, \{v \in \mathcal{V} \mid s(v) \leq \tau < f(v)\,\} \,\Big|,$$

kar je število v trenutku τ izvršujočih se točk GPP oz. število procesorjev, ki so zasedeni v trenutku τ. Število procesorjev, ki so potrebni za nemoteno izvršitev GPP v času T, kot to narekuje sprožitvena funkcija s, je tedaj

$$p_{T,s} = \max_{0 \leq \tau \leq T} p_{T,s}(\tau).$$

Najmanjše število procesorjev, ki so potrebni za izvršitev GPP v času T pa je

$$p_T = \min_{s \in \mathcal{S}(T)} p_{T,s}.$$

Sprožitvene funkcije, ki minimizirajo zgornji izraz imenujemo *procesorsko optimalne* sprožitvene funkcije – okrajšano p-optimalne. Iskanje p-optimalnih sprožitvenih funkcij je smiselno le za $T \in [T_\infty, T_1]$. Posebej se bomo osredotočili na primer $T = T_\infty$, ki opisuje iskanje minimalnega števila procesorjev p_{T_∞} za najhitrejšo izvršitev GPP.

1.5.1 Ocene spodnje meje

Sedaj predpostavljamo, da je $T = T_\infty$. Kot smo že omenili je problem iskanja vrednosti p_{T_∞} NP-poln. Zato nadomestimo natančno računanje p-optimalne sprožitvene funkcije s hevristično konstrukcijo sprožitvene funkcije, ki pa je "zadosti blizu" p-optimalni. Pri hevrističnem konstruiranju sprožitvene funkcije je koristno poznati čim bolj natančno spodnjo mejo za p_{T_∞}. Zato bomo v nadaljevanju uvedli nekatere metode za računanje spodnje meje $\tilde{p}_{T_\infty}$. Kvaliteto metode določata: časovna zahtevnost računanja $\tilde{p}_{T_\infty}$ in natančnost $\tilde{p}_{T_\infty}$, ki jo določa vrednost $p_{T_\infty} - \tilde{p}_{T_\infty}$.

Fernandez - Bussellova ocena. Zelo natančno oceno za $\tilde{p}_{T_\infty}$ podajata *Fernandez* in *Bussell* v [1]. To oceno določimo na sledeči način. Naj bo v poljubna točka iz $GPP = (G, t)$. Interval $I(v) = [s_e(v), f_l(v)]$ imenujemo *interval izvrševanja* točke v. Za nekritične točke, torej točke $v \in \mathcal{V} - \mathcal{K}$, velja $t(v) < f_l(v) - s_e(v)$, kar pomeni, da se sme njihovo izvrševanje "premikati" znotraj $I(v)$. Naj bosta τ_1 in τ_2 poljubni naravni števili in naj velja $0 \leq \tau_1 < \tau_2 \leq T_\infty$. Interval $J(v, \tau_1, \tau_2) = I(v) \cap [\tau_1, \tau_2]$ imenujemo *interval prekrivanja* točke v glede na τ_1 in τ_2. Kadar je $J(v, \tau_1, \tau_2) \neq \emptyset$, smemo premakniti izvrševanje nekritične točke v tako, da minimiziramo čas njenega izvrševanja znotraj $J(v, \tau_1, \tau_2)$. Vsoto vseh tako minimiziranih časov, gledano po vseh $v \in \mathcal{V}$, označimo s $K(\tau_1, \tau_2)$. Končno smemo zapisati oceno

$$\tilde{p}_{T_\infty} = \max_{\tau_1 < \tau_2} \left\lceil \frac{K(\tau_1, \tau_2)}{\tau_2 - \tau_1} \right\rceil.$$

V nadaljevanju bomo to oceno označevali s $\tilde{p}_{T_\infty, FB}$. Računanje $\tilde{p}_{T_\infty, FB}$ zajema obravnavo $T_\infty(T_\infty + 1)/2$ intervalov $[\tau_1, \tau_2]$, kar pomeni, da je časovna zahtevnost reda $\mathcal{O}(T_\infty^2)$. Vidimo, da ima računanje $\tilde{p}_{T_\infty, FB}$ relativno veliko časovno zahtevnost, zato si bomo v nadaljevnaju ogledali še nekatere manj natančne toda hitrejše metode za izračun $\tilde{p}_{T_\infty}$.

Chen - Epleyeva ocena. V [2][4] je vpeljana ocena za p_{T_∞}, ki temelji na povprečni vzporednosti $S_\infty =$

[4] To oceno je dejansko prvi vpeljal McNaughton v [3]

$\frac{T_1}{T_\infty}$ in se glasi

$$\tilde{p}_{T_\infty} = \lceil S_\infty \rceil.$$

To oceno bomo označevali s $\tilde{p}_{T_\infty,CE}$.

Hujeva ocena. *Hu* je v [4] vpeljal oceno za p_{T_∞} na sledeč način. Naj bo $\mathcal{L}(\tau) = \{v \in \mathcal{V} \mid f_l(v) = \tau\}$, torej množica vseh točk, ki se morajo izstreliti najkasneje v trenutku τ. Potem je

$$\tilde{p}_{T_\infty} = \max_{0 \le \omega \le T_\infty} \left[\frac{1}{\omega} \sum_{\tau=1}^{\omega} \mid \mathcal{L}(\tau) \mid \right].$$

Hujevo oceno bomo označevali s $\tilde{p}_{T_\infty,H}$ in jo imenovali tudi *največja povprečna vzporednost.*

Ramamoorthy - Chandy - Gonzalezova ocena. *Ramamoorthy et al.* so v [5] izboljšali $\tilde{p}_{T_\infty,H}$ tako, da so upoštevali tudi *kritično vzporednost*, ki je vpeljana kot

$$\kappa = \max_{0 \le \tau \le T_\infty} \mid \mathcal{K}(\tau) \mid.$$

Oceno so definirali kot

$$\tilde{p}_{T_\infty} = \max(\tilde{p}_{T_\infty,H}, \kappa).$$

Njihovo oceno pa bomo označili s $\tilde{p}_{T_\infty,R}$.

Razširjena kritična vzporednost. Razvidno je, da je kritična vzporednost κ precej šibka ocena za p_{T_∞}, saj se pri relativno velikih grafih njen vpliv izniči. Takrat postane $\tilde{p}_{T_\infty,R} \approx \tilde{p}_{T_\infty,H}$. Eden od načinov za izboljšanje natančnosti ocene je torej izostritev kritične vzporednosti κ, kar bomo storili v nadaljevanju. Predpostavimo, da je v vseh trenutkih τ intervala $[\tau_1, \tau_2]$ v vsaki množici $\mathcal{K}(\tau)$ po κ elementov. Zgodi se, da se mora v $[\tau_1, \tau_2]$ poleg kritičnih izvrševati tudi katera od nekritičnih točk. Kadar obstaja vsaj ena taka točka, potrebujemo dodatni procesni element, torej je potrebnih $\kappa + 1$ procesorjev! Do podobne situacije pa seveda pogosto pride tudi v intervalih, kjer je število kritičnih točk sicer manjše od κ, vendar se mora znotraj tega intervala (delno) izvrševati toliko nekritičnih točk, da skupno število potrebnih procesorjev preseže κ. V nadaljevanju bomo zato vpeljali *razširjeno kritično vzporednost* κ' in jo uporabili pri lastni oceni za p_{T_∞} [6].

Na končnem intervalu $[0, T_\infty]$ poiščimo najmanjše končno zaporedje trenutkov $0 = \tau_0 < \tau_1 < \ldots < \tau_k = T_\infty$, in sicer takšno, da je na vsakem intervalu $[\tau_{j-1}, \tau_j)$, $j = 1, 2, \ldots, k$ število kritičnih točk $\mid \mathcal{K}(\tau) \mid$ stalno. To število krajše označimo s κ_{j-1}. Definirajmo množico $\mathcal{W}_{j-1}$ tistih nekritičnih točk, ki zahtevajo v intervalu $[\tau_{j-1}, \tau_j)$ procesor vsaj za en trenutek:

$$\begin{aligned} \mathcal{W}_{j-1} = \{ \; & v \in \mathcal{V} - \mathcal{K} \mid (s_e(v) \ge \tau_{j-1} \lor \\ & f_e(v) > \tau_{j-1} > s_e(v)) \land \\ & (f_l(v) \le \tau_j \lor f_l(v) > \tau_j > s_l(v)) \, \}. \end{aligned}$$

Izrševanje točke $v \in \mathcal{W}_{j-1}$ se časovno prekriva (v vsaj enem trenutku) z izvrševanjem kritičnih točk v časovnem intervalu $[\tau_{j-1}, \tau_j)$. To prekrivanje je v splošnem krajše od časa $t(v)$, kajti točka se lahko sproži že pred trenutkom τ_{j-1} in/ali konča izvrševanje po trenutku τ_j. Zato upoštevamo pri izračunu skupnega prekrivanja v intervalu $[\tau_{j-1}, \tau_j)$ za vsako točko $v \in \mathcal{W}_{j-1}$ le njeno minimalno prekrivanje $\omega_{j-1}(v)$, ki je

$$\omega_{j-1}(v) = \min\{f_e(v) - \tau_{j-1}, \; \tau_j - s_l(v), \; t(v)\}$$

Minimalno potrebno število dodatnih procesnih elementov v časovnem intervalu $[\tau_{j-1}, \tau_j)$ je tako:

$$\begin{aligned} \lambda_{j-1} = & \Big(\sum_{v \in \mathcal{W}_{j-1}} \omega_{j-1}(v) \Big) \Big/ \\ & \max\{\tau_{j-1}, \min_{v \in \mathcal{W}_{j-1}} s_e(v)\} - \min\{\tau_j, \max_{v \in \mathcal{W}_{j-1}} f_l(v)\}. \end{aligned}$$

Končno je mogoče zapisati razširjeno kritično vzporednost κ' kot

$$\kappa' = \max_{1 \le j \le k} \lceil (\kappa_j + \lambda_j) \rceil$$

in definiramo oceno za p_{T_∞} takole:

$$\tilde{p}_{T_\infty} = \max(\tilde{p}_{T_\infty,H}, \kappa').$$

To oceno bomo označili tudi s $\tilde{p}_{T_\infty,\kappa'}$.

Opisana metoda je poenostavitev Fernandez-Bussellove metode, v kateri namesto $\mathcal{O}(T_\infty^2)$ intervalov analiziramo največ $\mathcal{O}(T_\infty)$ disjunktnih intervalov. S takšno poenostavitvijo pa se natančnost ocene ne poslabša bistveno.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.85897	.91815	.93984	.98225
21 - 40	.81865	.89119	.89983	.93207
41 - 60	.87800	.91306	.91595	.92843
61 - 80	.88899	.91847	.92151	.93322
81 - 100	.88649	.91497	.91904	.92596
1 - 100	.87006	.91143	.91768	.93529

Tabela 1: Relativna natančnost ocen pri $t(v) = 1$.

Primerjava ocen. Ocene za spodnjo mejo potrebnega števila procesorjev smo računali[5]

[5] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.88188	.92392	.92392	.98498
21 - 40	.82895	.89294	.89354	.94139
41 - 60	.86430	.90486	.90486	.91888
61 - 80	.89379	.91816	.91816	.93241
81 - 100	.89224	.91446	.91446	.92759
1 - 100	.87384	.91040	.91051	.93561

Tabela 2: Relativna natančnost ocen pri $t(v) \leq 5$.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.88200	.92100	.92100	.98200
21 - 40	.82912	.89350	.89350	.93622
41 - 60	.87045	.90739	.90739	.92385
61 - 80	.89447	.91911	.91911	.93026
81 - 100	.89115	.91370	.91370	.92194
1 - 100	.87510	.91072	.91072	.93348

Tabela 3: Relativna natančnost ocen pri $t(v) \leq 10$.

po Fernandez-Bussellovi, Chen-Epleyevi, Hujevi, Ramamoorthy-Chandy-Gonzalezovi in lastni metodi. Skupno smo analizirali 7.500 naključno generiranih grafov pretoka podatkov. Pri tem smo spreminjali število točk $n \leq 100$ in čas njihovega izvrševanja $t(v) \leq 10$. Rezultati analiz so prikazani v tabelah 1, 2 in 3. Natančnost metod smo določali glede na najnatančnejšo, tj. Fernandez-Bussellovo oceno $\tilde{p}_{T_\infty,FB}$, saj je število p_{T_∞} neznano. Rezultati potrjujejo, da je naša metoda po svoji natančnosti najbliže $\tilde{p}_{T_\infty,FB}$, saj je v povprečju le za 6.52% slabša, hkrati pa je za red velikosti hitrejša. Naši oceni sledijo $\tilde{p}_{T_\infty,R}$ (8.70%), $\tilde{p}_{T_\infty,H}$ (8.92%) in $\tilde{p}_{T_\infty,CE}$ (12.71%). Primerjava med $\tilde{p}_{T_\infty,H}$ in $\tilde{p}_{T_\infty,R}$ kaže, da je slednja ocena boljša le pri tistih GPP, katerih vrednosti $t(v)$ se spreminjajo zelo malo. Samo v takšnih grafih kritična vzporednost prevlada nad največjo povprečno vzporednostjo. Razširjena kritična vzporednost, ki smo jo vpeljali pri svoji oceni $\tilde{p}_{T_\infty,\kappa'}$, pa prevlada tudi v tistih GPP, kjer je $t(v)$ spremenljiv.

1.6 Zahteve po času

Naj bosta dana $GPP = (G, t)$ in naravno število $p \leq p_{T_\infty}$. Iščemo najmanjše naravno število $T_p \geq T_\infty$, za katero je množica sprožitvenih funkcij $\mathcal{S}(T_p) \neq \emptyset$. Posledica pomanjkanja procesorjev je *podaljšanje časa izvrševanja* ΔT_p, ki znaša

$$\Delta T_p = T_p - T_\infty.$$

Sprožitvene funkcije, ki minimizirajo zgornji izraz imenujemo *časovno optimalne* sprožitvene funkcije – krajše T-optimalne. Iskanje T-optimalnih sprožitvenih funkcij je seveda smiselno le za $p \leq p_{T_\infty}$.

1.6.1 Ocene spodnje meje

Tudi problem iskanja vrednosti T_p je NP-poln, zato namesto natančnega računanja T-optimalne sprožitvene funkcije uporabimo njeno hevristično konstrukcijo. Podobno kot pri iskanju p-optimalnih sprožitvenih funkcij je tudi tu koristno poznati kar se da natančno spodnjo mejo za T_p, v nadaljevanju označeno s $\tilde{T}_p$. Tudi tokrat določata kvaliteto metode za računanje spodnje meje: časovna zahtevnost računanja $\tilde{T}_p$ in natančnost $\tilde{T}_p$, ki jo določa vrednost $T_p - \tilde{T}_p$.

Hujeva ocena. *Hu* je v [4] vpeljal za T_p sledečo oceno: Naj bo zopet $\mathcal{L}(\tau)$ množica vseh točk, ki se morajo izstreliti najkasneje v trenutku τ. Potem je

$$\tilde{T}_p = T_\infty + \max_{0 \leq \omega \leq T_\infty} \left[-\omega + \frac{1}{p} \sum_{\tau=1}^{\omega} | \mathcal{L}(\tau) | \right].$$

Hujevo oceno bomo označevali s $\tilde{T}_{p,H}$.

Fernandez - Bussellova ocena. Izboljšanje ocene $\tilde{T}_{p,H}$ sta opisala *Fernandez* in *Bussell* v [1]. To oceno določimo s sredstvi, ki so bila vpeljana med konstruiranjem $\tilde{p}_{T_\infty,FB}$. Naj bosta τ_1 in τ_2 poljubni naravni števili in naj velja $0 \leq \tau_1 < \tau_2 \leq T_\infty$. Če je na voljo le p procesorjev, potem se izvrševanje točk znotraj intervala $[\tau_1, \tau_2]$ podaljša za najmanj $\frac{K(\tau_1,\tau_2)}{p} - (\tau_2 - \tau_1)$. Od tod moremo določiti oceno za $\tilde{T}_p$, ki je

$$\tilde{T}_p = T_\infty + \max_{\tau_1 < \tau_2} \left[\frac{K(\tau_1, \tau_2)}{p} - (\tau_2 - \tau_1) \right].$$

V nadaljevanju bomo to oceno označevali s $\tilde{T}_{p,FB}$.

Primerjava ocen. V [1] je pokazano, da je Fernandez-Bussellova ocena natančnejša od Hujeve. Ta natančnost se doseže za ceno večje časovne kompleksnosti, ki znaša $\mathcal{O}(T_\infty^2)$ in je za razred večja od Hujeve.

2. Mehanizmi sinhronizacije

Sedaj bomo opisali hevristična algoritma za konstrukcijo T- in p-optimalnih sprožitvenih funkcij, ki ju bomo uporabili v dveh hevrističnih algoritmih za dodeljevanje; z njima dani GPP porazdelimo med procesorje [9].

vhod: GPP, p.
izhod: $SGPP(p, T_p)$, oz. $s(v)$, za vsak $v \in \mathcal{V}$.

```
τ := 0;  T_p := 0;  q := 0;  W := V
repeat
  if q > 0 then
    V_p := {v ∈ V | f(v) = τ};   W := W − V_p;   q := q − |V_p|
  endif
  V_i := {v ∈ V | v ima vse vhodne podatke};
  -- Množica izvršljivih točk je V_i = K(τ) ∪ V_n, kjer sta K(τ) in V_n
  -- množici kritičnih oz. nekritičnih točk na globini τ.
  if q < p then
    if p − q ≤ |K(τ)| then bodi V_d ⊂ K(τ), kjer je |V_d| ≤ p − q else
    if p − q ≥ |V_i| then V_d := V_i else
      Bodi V_d ⊂ V_n, kjer je |V_d| ≤ p − q − |K(τ)|;  V_d := V_d ∪ K(τ)
    endif
    q := q + |V_d|          -- Sproži se nekaj dodatnih, naključno izbranih točk.
  endif
  forall v ∈ V_d do s(v) := τ endforall
  T_p := τ;  τ := τ + 1
until W = ∅;
```

Algoritem TOptSinh: Konstruiranje T-optimalne sprožitvene funkcije.

2.1 Sinhronizacija

Algoritma za konstrukcijo optimalnih sprožitvenih funkcij, ki ju bomo opisali v tem poglavju, temeljita na spoznanju, da si moremo izvrševanje GPP predočiti s pomočjo prehajanja točk iz ene množice (stanja) v drugo. Že v prej smo vpeljali eno takšnih množic – množico $\mathcal{K}(\tau)$ kritičnih točk v trenutku τ. Nadalje za vsak trenutek τ vpeljemo še množici: $\mathcal{V}_i$ *izvršljivih* točk, tj. točk, ki so zbrale vse vhodne podatke, a se še niso sprožile in $\mathcal{V}_p = \{v \in \mathcal{V} | f(v) = \tau\}$ *pozabljenih* točk, tj. točk, ki so se izstrelile v tem trenutku. V splošnem so poleg kritičnih v množici $\mathcal{V}_i$ tudi točke, ki smejo svojo sprožitev odložiti, ne da bi to nujno vplivalo na najkrajše izvrševanje GPP. Te točke imenujemo *nekritične* v trenutku τ in jih zberemo v množico $\mathcal{V}_n$.

Par (GPP, s), kjer je s sprožitvena funkcija, ki posredno določa tudi p in T, imenujemo *sinhronizirani* graf pretoka podatkov ter ga označimo z $SGPP(p, T)$. Torej T-optimalna sprožitvena funkcija določa $SGPP(p, T_p)$, medtem ko p-optimalna funkcija določa $SGPP(p_{T_\infty}, T_\infty)$.

2.1.1 T-optimalna sprožitvena funkcija

Za konstruiranje T-optimalne sprožitvene funkcije smo razvili algoritem, ki smo ga imenovali TOptSinh. Ta nam za dani GPP ter vnaprej določeno število p procesorjev vrne takšno sprožitveno funkcijo s, ki zagotavlja čas izvrševanja T_p, ki je kar se da blizu oceni $\tilde{T}_p$.

Kvaliteto algoritma smo ocenjevali z odstotkom primerov, ko je čas T_p dosegel oceno $\tilde{T}_{p,H}$, saj natančne vrednosti T_p ne poznamo. Algoritem je bil preverjen[6] nad 500 GPP in je v 75.6 % dosegel $T_p = \tilde{T}_{p,H}$. Obenem smo merili tudi upad idealne pospešitve D_p v odvisnosti od pomanjkanja procesorjev. Primerjava med asinhronim podatkovno vodenim računanjem (sprožitvena funkcija s_e) in sinhroniziranim podatkovno vodenim računanjem (T-optimalna sprožitvena funkcija) je prikazana v tabeli 4, kjer so podani povprečni rezulati.

Ključni del algoritma TOptSinh je v konstruiranju množice $\mathcal{V}_d$, tj. v sprožanju dodatnih izvršljivih točk, kadar so na voljo prosti procesorji. Absolutno prednost pri sprožanju imajo kritične točke v danem trenutku (zaradi omejenega števila procesorjev seveda v splošnem pride do odloga sprožitve celo pri nekaterih kritičnih točkah). Uvrščanje nekritične točk v množico $\mathcal{V}_d$ pa smo izvajali v skladu z naslednjimi strategijami: naključno izbiranje (v vrstnem redu, kot prihajajo

[6] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

vhod: GPP, T_∞.
izhod: $SGPP(p_{T_\infty}, T_\infty)$, oz. $s(v)$, za vsak $v \in \mathcal{V}$.

Izračunaj $\tilde{p}_{T_\infty}$; $\tau := 0$; $p_{T_\infty} := 0$; $q := 0$
repeat
  if $q > 0$ **then**
    $\mathcal{V}_p := \{v \in V | f(v) = \tau\}$; $q := q - |\mathcal{V}_p|$
  endif
  $\mathcal{V}_i := \{v \in \mathcal{V} | v$ ima vse vhodne podatke$\}$;
  — Množica izvršljivih točk je $\mathcal{V}_i = \mathcal{K}(\tau) \cup \mathcal{V}_n$, kjer sta $\mathcal{K}(\tau)$ in $\mathcal{V}_n$
  — množici kritičnih oz. nekritičnih točk na globini τ.
  $q := q + |\mathcal{K}(\tau)|$ — Sprožijo se vse kritične točke.
  if $q < \tilde{p}_{T_\infty}$ **then**
    Bodi $\mathcal{V}_d \subset \mathcal{V}_n$, kjer je $|\mathcal{V}_d| \le \tilde{p}_{T_\infty} - q$;
    $q := q + |\mathcal{V}_d|$ — Sproži se nekaj dodatnih, naključno izbranih točk.
  endif
  forall $v \in \mathcal{K}(\tau) \cup \mathcal{V}_d$ **do** $s(v) := \tau$ **endforall**
  $p_{T_\infty} := \max(q, p_{T_\infty})$; $\tau := \tau + 1$
until $\tau = T_\infty$;

Algoritem pOptSinh: Konstruiranje p-optimalne sprožitvene funkcije.

p_{T_∞}	$p = 3/4 p_{T_\infty}$	$p = 1/2 p_{T_\infty}$	$p = 1/4 p_{T_\infty}$
Sprožitvena funkcija s_e			
4	0.030	0.223	1.232
5	0.016	0.121	0.454
6	0.006	0.205	0.621
7	0.004	0.104	0.761
8	0.015	0.147	0.883
9	0.006	0.071	0.397
10	0.003	0.100	0.464
$\sum$	0.011	0.139	0.687
T-optimalna sprožitvena funkcija			
4	0.011	0.211	1.079
5	0.002	0.048	0.375
6	0.002	0.117	0.537
7	0.000	0.029	0.718
8	0.000	0.044	0.789
9	0.000	0.001	0.324
10	0.000	0.018	0.311
$\sum$	0.002	0.067	0.590

Tabela 4: Upad idealne pospešitve D_p.

v $\mathcal{V}_i$), po naraščajočih $t(v)$ in po padajočih $t(v)$. Poskusi so pokazali, da nobena od strategij ni bila izrazito boljša od ostalih.

2.1.2 p-optimalna sprožitvena funkcija

Algoritem pOptSinh za dani GPP ter pripadajoči najkrajši čas izvrševanja T_∞ vrne sprožitveno funkcijo s, ki zagotavlja izvrševanje GPP na številu procesorjev, ki je kar se da blizu ocene $\tilde{p}_{T_\infty}$. Algoritem v vsakem trenutku τ poskuša sprožiti čimveč, toda kvečjemu $\tilde{p}_{T_\infty}$ izvršljivih točk. V vsakem trenutku τ sprožimo vse točke iz $\mathcal{K}(\tau)$. Če je zasedenih procesorjev še vedno manj kot $\tilde{p}_{T_\infty}$, na njih sprožimo nekatere nekritične točke, ki jih izberemo naključno (v vrstnem redu, kot prihajajo v $\mathcal{V}_i$). Iz povedanega je očitno, da je pri tem odločilna natančnost $\tilde{p}_{T_\infty}$.

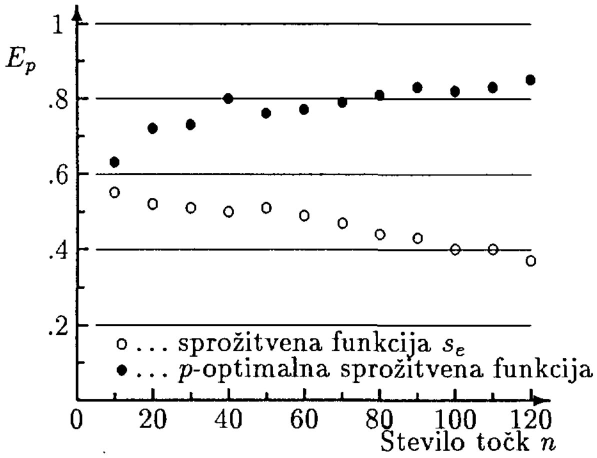
Optimalnost algoritma smo ocenjevali z odstotkom primerov, ko je p_{T_∞} dosegel oceno $\tilde{p}_{T_\infty,R}$, $\tilde{p}_{T_\infty,\kappa'}$ oziroma $\tilde{p}_{T_\infty,FB}$, saj natančne vrednosti p_{T_∞} ne poznamo. Algoritem je bil preverjen[7] nad 1.500 GPP in je dosegel rezultate, prikazane v tabeli 5.

$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$	$\tilde{p}_{T_\infty,FB}$
0.705	0.783	0.824

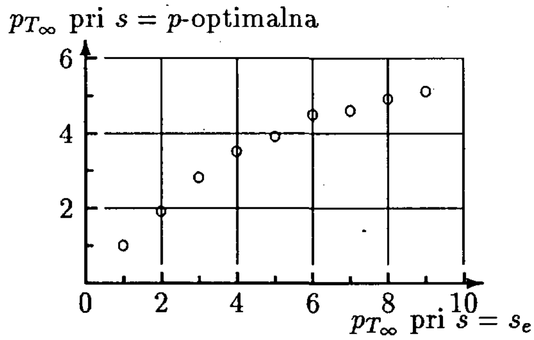
Tabela 5: Optimalnost algoritma pOptSinh.

Pri problemu optimizacije procesorjev je ključen podatek izkoriščenost procesorjev E_p, za katero želimo, da se čimbolj približa vrednosti 1. Slika

[7] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.



Slika 1: Izkoriščenost procesorjev E_p.



Slika 2: Potrebe po procesorjih.

1 prikazuje izkoriščenost procesorjev E_p v odvisnosti od velikosti GPP za asinhro podatkovno vodeno računanje (sprožitvena funkcija s_e) in sinhronizirano podatkovno vodeno računanje (p-optimalna sprožitvena funkcija). Računanje, ki sledi p-optimalni sprožitveni funkciji, ima mnogo manjše zahteve po procesorjih kot računanje, ki sledi s_e sprožitveni funkciji, kar potrjujejo rezultati, prikazani na sliki 2. Tudi tu smo analizirali 1.500 GPP, pri čemer smo spreminjali število točk $n \leq 120$ in čas njihovega izvrševanja $t(v) \leq 10$.

2.2 Dodeljevanje

Po končani fazi sinhroniziranja je GPP pridružena sprožitvena funkcija s in skupaj tvorita $SGPP(p,T)$. S tem so vse $v \in \mathcal{V}$ urejene po trenutkih sprožitve $s(v)$. Takšna urejenost zagotavlja, da se GPP izvrši na p procesorjih v času T ob uporabi najsplošnejšega algoritma za dodeljevanje (algoritem `NakljDod`) [7], ki točko v dodeli *naključno izbranemu* prostemu procesorju.

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in \mathcal{V}$.

Razvrsti pare $(v, s(v))$ po naraščajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := 0$ **endfor**
repeat
  Vzemi iz S vse pare z enakim s in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \leq s\}$
  forall $v \in \mathcal{W}$ **do**
    Naključno izberi $q \in P$.
    $\pi(v) := q$;
    $F[q] := f(v)$; $P := P - \{q\}$
  endforall
  $W := \emptyset$; $P := \emptyset$
until $S = \emptyset$;

Algoritem `NakljDod`: Naključno dodeljevanje.

Po končanem dodeljevanju je vsaki točki $v \in \mathcal{V}$ pridružen procesor z *indeksom* $\pi(v)$, kjer je $1 \leq \pi(v) \leq p$. V vsakem trenutku je na voljo vsaj toliko prostih procesorjev, kot je točk, ki se morajo tedaj sprožiti. Dodeljevanje teh točk je poljubno, kar pomeni, da moremo v splošnem dani $SGPP(p,T)$ porazdeliti na več načinov. Vse dodelitve so enakovredne, saj vse zagotavljajo, da se GPP izvrši na p procesorjih v času T.

Točki u in v iz $\mathcal{V}$ sta *ločeni*, če sta dodeljeni različnima procesorjema. Povezavo med ločenima točkama imenujemo *globalna* povezava. V primeru hibridne arhitekture poteka komunikacija med ločenima točkama preko nadzorno-povezovalne enote in v splošnem zahteva čas $t_c > 0$. V takšnem primeru najsplošnejši algoritem torej ne vrača več enakovrednih dodelitev, saj se število globalnih povezav spreminja. Že prej smo videli, da število globalnih povezav vpliva na čas izvrševanja GPP. Zato se bomo osredotočili na konstruiranje takšnih algoritmov za dodeljevanje, ki bodo minimizirali število globalnih povezav v dodelitvah [8]. Točneje: trivialni kriterij naključnega dodeljevanja prostih procesorjev bomo nadomestili s kompleksnejšim, ki se glasi:

Dodeli točke w množice $\mathcal{W}$ procesorjem q množice P tako, da bo $\sum c(w,q)$ maksimalna, kjer je $c(w,q)$ število sosednjih točk točke w, ki so bile doslej dodeljene procesorju q.

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in \mathcal{V}$.

Razvrsti pare $(v, s(v))$ po naraščajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := 0$ **endfor**
repeat
  Vzemi iz S vse pare z enakim s in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \leq s\}$
  forall $v \in \mathcal{W}$ **do**
    forall $q \in P$ **do**
      $c(v,q)$ = število neposrednih predhodnikov od v, ki so bili dodeljeni v q-ti procesor.
    endforall
  endforall
  Reši problem WBM za graf $(\mathcal{W} \cup P, \mathcal{W} \times P)$.
  forall pare (v,q), ki so del rešitve **do**
    $\pi(v) := q$;
    $F[q] := f(v)$
  endforall
  $\mathcal{W} := \emptyset$; $P := \emptyset$
until $S = \emptyset$;

Algoritem `MinGlDol`: Minimizacija globalnih povezav (navzdol).

Dodeljevanje, ki poteka v skladu z zgornjim pravilom, je v nekem smislu konservativno, saj poskuša točko pridružiti procesorju, v katerem je največ njenih sosedov. Opisano pravilo je primer znanega problema UTEŽENEGA DVODELNEGA UJEMANJA (WBM[8]) [10, 11]:

Naj bo dan dvodelni graf $G = (\mathcal{W} \cup P, E)$, kjer je $E \subseteq \mathcal{W} \times P$ ter cenovna funkcija $c : E \longrightarrow \mathbb{N}$. Iščemo ujemanje $M \subseteq E$ (množico povezav M, v kateri noben par povezav nima skupne točke) tako, da je $\sum_{(w,q)\in M} c(w,q)$ maksimalna.

Problem WBM znamo rešiti v času $\mathcal{O}(ne\log n/\max(1,\log\frac{e}{n}))$, kjer je $e = \mid E \mid$ in $n = \mid \mathcal{W} \mid + \mid P \mid$.

V algoritmu `MinGlDol` poteka dodeljevanje od začetnih točk proti končnim, medtem ko poteka dodeljevanje v algoritmu `MinGlGor` v nasprotni smeri. V obeh algoritmih se pojavlja problem WBM, pri čemer v prvem primeru določajo ceno $c(w,q)$ neposredni predhodniki točke w, v drugem pa njeni neposredni nasledniki.

2.3 Primeri

Z nekaterimi primeri bomo pokazali učinkovitost časovne optimizacije asinhronega računanja s pomočjo mehanizmov sinhronizacije. Obravnavali bomo naslednje primere: hitro Fourierjevo transformacijo (FFT), dinamično analizo scene (DAS) in LU razcep matrike (LU). Privzeli bomo hibridno arhitekturo s $p = 3$ procesorji (P_1, P_2 in P_3), katere medprocesorsko komunikacijo t_c bomo ustrezno spreminjali. Kvaliteto naših rezultatov, ki jih dobimo z uporabo sinhronizacijskega algoritma `TOptSinh` ter dodeljevalnih algoritmov `MinGlDol` in `MinGlGor`, bomo primerjali z naslednjimi znanimi razvrščevalnimi algoritmi: `CPM`[9] [12], `HNF`[10] [13] in `WL`[11] [13].

2.3.1 FFT: Hitra Fourierjeva transformacija

Povrnimo se na primer izračuna hitre Fourierjeve transformacije na 8 točkah, ki smo ga že obravnavali v prvem delu. Predpostavimo, da se točke A, C, E, G, I, J, M, N, Q, R, S in T izvršujejo eno časovno enoto, točke B, D, F, H, K, L, O, P, U, V, W in X pa pet časovnih enot.

`CPM`, `HNF` in `WL` algoritmi. Za primer FFT algoritma dajejo vse tri metode (`CPM`, `HNF`, `WL`)

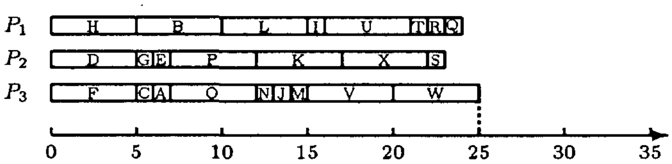
[8] Weighted Bipartite Matching

[9] Critical Path Method
[10] Heavy Node First
[11] Weighted Length

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in V$.

Razvrsti pare $(v, s(v))$ po padajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := T$ **endfor**
repeat
  Vzemi iz S vse pare z enakim f in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \geq f\}$;
  forall $v \in \mathcal{W}$ **do**
    forall $q \in P$ **do**
      $c(v,q)$ = število neposrednih naslednikov od v, ki so bili dodeljeni v q-ti procesor.
    endforall
  endforall
  Reši problem WBM za graf $(\mathcal{W} \cup P, \mathcal{W} \times P)$.
  forall pare (v,q), ki so del rešitve **do**
    $\pi(v) := q$
    $F[q] := s(v)$;
  endforall
  $\mathcal{W} := \emptyset$; $P := \emptyset$;
until $S = \emptyset$

Algoritem `MinGlGor`: Minimizacija globalnih povezav (navzgor).



Slika 3: FFT: Razvrstitev po `CPM`, `HNF` in `WL`.

v	$s(v)$	v	$s(v)$	v	$s(v)$
A	6	I	15	Q	23
B	5	J	13	R	22
C	5	K	12	S	22
D	0	L	10	T	21
E	6	M	14	U	16
F	0	N	12	V	15
G	5	O	7	W	20
H	0	P	7	X	17

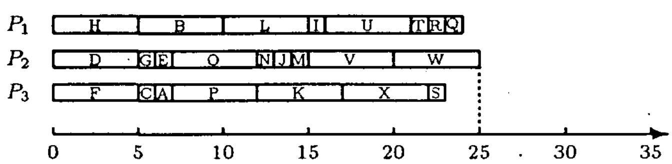
Tabela 7: FFT: T-optimalna sprožitvena funkcija.

enakovredne rezultate. Točke se dodelijo procesorjem, kot je prikazano na sliki 3. Razvidno je, da dobimo v tem primeru 22 globalnih povezav. Upad idealne pospešitve D_p se pri $t_c > 0$ poveča, tako dobimo pri $t_c = 10$ čas izvrševanja $T_p = 43$, kar se odraža tudi v povečanju upada idealne pospešitve $D_p = 1.87$. Podrobnejši rezultati `CPM`, `HNF` in `WL` dodeljevanja v odvisnosti od časa t_c so prikazani v tabeli 6.

t_c	T_p	S_p	E_p	D_p
0	25	2.88	0.96	0.67
2	27	2.67	0.89	0.80
4	31	2.32	0.77	1.07
10	43	1.67	0.56	1.87

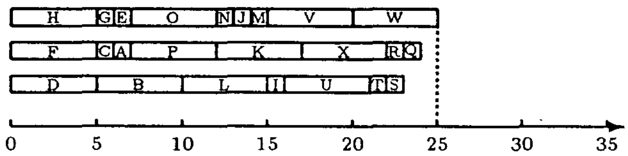
Tabela 6: FFT: `CPM`, `HNF` in `WL` dodeljevanje pri različnih t_c.

`MinGlDol` in `MinGlGor` algoritma. *GPP* FFT algoritma najprej sinhroniziramo, za kar uporabimo algoritem `TOptSinh`, ki vrne T-optimalno sprožitveno funkcijo. Rezultat je prikazan v tabeli 7.



Slika 4: FFT: Razvrstitev po `MinGlDol`.

V drugem koraku uporabimo dodeljevalni algoritem `MinGlDol` oz. `MinGlGor`. Razvrstitev točk po `MinGlDol` algoritmu je prikazan na sliki 4, medtem ko je razvrstitev po `MinGlGor` algoritmu prikazana na sliki 5.



Slika 5: FFT: Razvrstitev po MinGlGor.

Z uporabo algoritmov MinGlDol in MinGlGor dobimo pri $t_c = 0$ enake rezultate kot pri algoritmih CPM, HNF in WL ($T_p = 25$, $S_p = 2.88$, $D_p = 0.667$ in $E_p = 0.96$). Pri $t_c > 0$ pa naša algoritma vračata rezultate z manjšim upadom idealne pospešitve D_p. Tako je pri $t_c = 10$ čas izvrševanja $T_p = 40$ (v prejšnjem primeru 43), kar pomeni, da je D_p le 1.67. Podrobnejši rezultati MinGlDol in MinGlGor dodeljevanja v odvisnosti od časa t_c so prikazani v tabeli 8.

t_c	T_p	S_p	E_p	D_p
0	25	2.88	0.96	0.67
2	25	2.88	0.96	0.67
4	28	2.57	0.86	0.87
10	40	1.80	0.60	1.67

Tabela 8: FFT: MinGlDol in MinGlGor dodeljevanje pri različnih t_c.

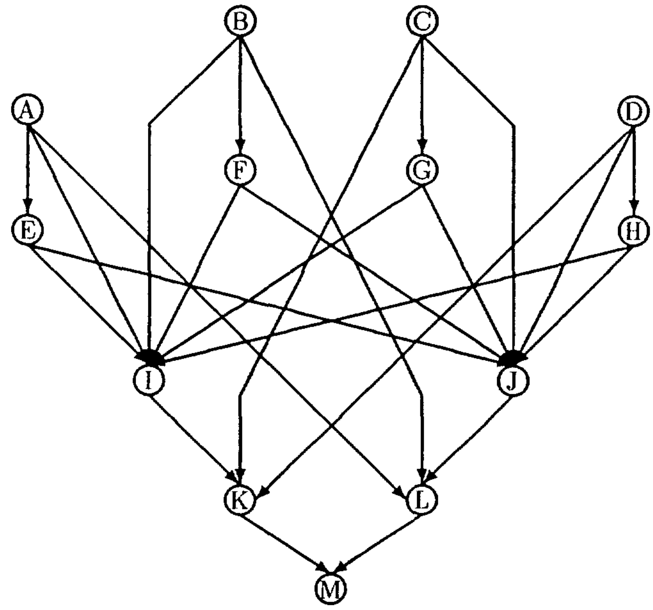
Oba algoritma občutno zmanjšata število globalnih povezav, tako dobimo pri MinGlDol algoritmu 16 globalnih povezav, algoritem MinGlGor pa nam število globalnih povezav zmanjša celo na 15.

2.3.2 DAS: Dinamična analiza scene

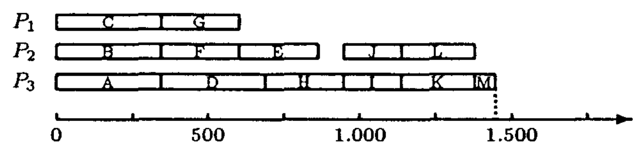
Za naslednji primer vzemimo algoritem za dinamično analizo scene. Algoritem DAS se uporablja pri izločanju podob gibljivih objektov in je podrobneje opisan v [14]. Pripadajoči *GPP* prikazuje slika 6. Točke A, B, C in D se izvršujejo 345 časovnih enot, točke E, F, G in H 259 časovnih enot, točki I in J 190 časovnih enot, točki K in L 241 časovnih enot in točka M 69 časovnih enot.

CPM, HNF in WL algoritmi. Tudi v primeru DAS algoritma dajejo metode (CPM, HNF in WL) enakovredne rezultate (slika 7).

MinGlDol in MinGlGor algoritma. Oba algoritma vračata enake rezultate kot algoritmi CPM, HNF in WL, torej pri $t_c = 0$ dobimo $T_p = 1449$, $S_p = 2.31$, $D_p = 0.31$ in $E_p = 0.77$ ter 12 globalnih povezav. Z algoritmom MinGlGor dobimo tudi enako dodelitev med procesorje (slika 7), medtem ko dobimo z al-



Slika 6: *GPP* DAS algoritma.



Slika 7: DAS: Razvrstitev po CPM, HNF, WL in MinGlGor.

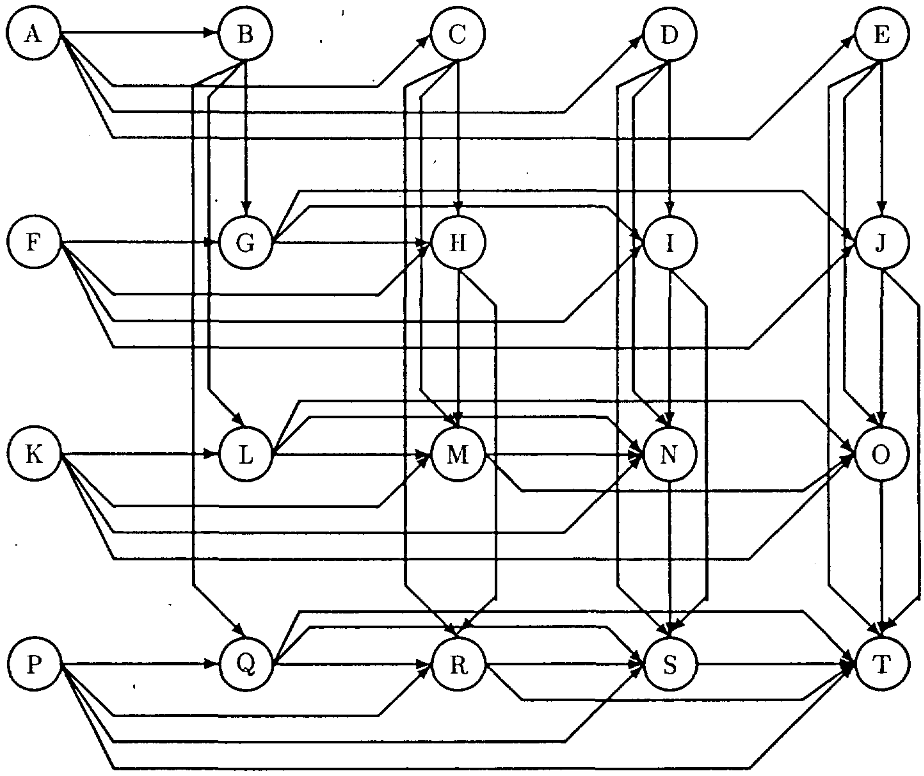
goritmom MinGlDol sicer drugačno dodelitev med procesorje (slika 8), ki pa, kot rečeno, rezultira v enako število globalnih povezev. Podrobnejša analiza pri $0 \leq t_c \leq 300$ je prikazana v tabeli 9.

t_c	T_p	S_p	E_p	D_p
0	1449	2.31	0.77	0.31
100	1649	2.03	0.68	0.49
200	1849	1.81	0.60	0.68
300	2049	1.63	0.54	0.86

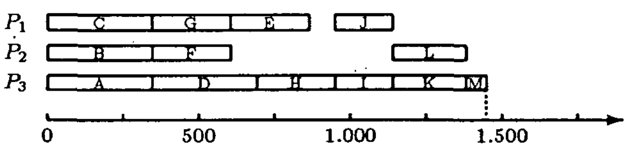
Tabela 9: DAS: CPM, HNF, WL, MinGlGor in MinGlDol dodeljevanje pri različnih t_c.

2.3.3 LU: LU razcep matrike

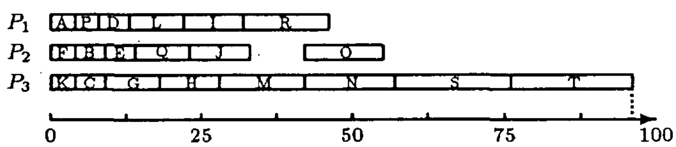
Za zadnji primer vzemimo algoritem za LU razcep matrike [15], ki je eden od najpogosteje uporabljenih algoritmov za reševanje sistema linearnih enačb. *GPP* LU algoritma je podan na sliki 9. Točke A, F, K in P se izvršujejo 4 časovne enote, točke B, C, D in E 5 časovnih enot, točke G, L in Q 9 časovnih enot, točke H, I in J 10 časovnih enot,



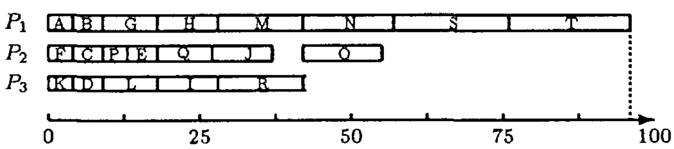
Slika 9: *GPP* algoritma za LU razcep matrike.



Slika 8: DAS: Razvrstitev po `MinGlDol`.



Slika 11: LU: Razvrstitev po `HNF`.



Slika 10: LU: Razvrstitev po `CPM` in `WL`.

točka O 13 časovnih enot, točki M in R 14 časovnih enot, točka N 15 časovnih enot, točka S 19 časovnih enot in točka T 20 časovnih enot.

`CPM`, `HNF` in `WL` algoritmi. LU razcep matrike je primer, da `HNF` algoritem vrača boljše rezultate od ostalih dveh. Razvrstitev točk po `CPM` oz. `WL` metodi je prikazana na sliki 10, medtem ko razvrstitev po `HNF` metodi na sliki 11. Pri analizi LU algoritma smo t_c spreminjali med 0 in 30 (tabeli 10 in 11). Metodi `CPM` in `WL` sta vrnili 36 globalnih povezav, medtem ko je `HNF` vrnila le 32 globalnih povezav. Zato je upad idealne pospešitve D_p pri $t_c = 10$ za 12.5% boljši.

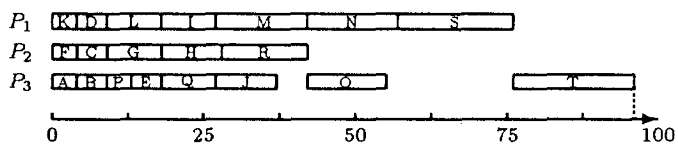
t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	126	1.49	0.50	0.45
20	166	1.13	0.38	0.91
30	206	0.91	0.30	1.37

Tabela 10: LU: `CPM` in `WL` dodeljevanje pri različnih t_c.

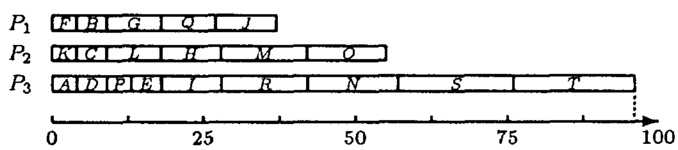
t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	122	1.54	0.51	0.40
20	162	1.16	0.39	0.86
30	202	0.93	0.31	1.32

Tabela 11: LU: HNF dodeljevánje pri različnih t_c.

MinGlDol in MinGlGor algoritma. Še boljše rezultate kot HNF pa vračata naša algoritma MinGlDol in MinGlGor. Razvrstitvi točk po MinGlDol ter MinGlGor sta opisani na slikah 12 in 13.



Slika 12: LU: Razvrstitev po MinGlDol.



Slika 13: LU: Razvrstitev po MinGlGor.

Pri $t_c \geq 0$ dosežeta obe metodi enak upad idealne pospešitve, ki je boljši od upada idealnih pospešitev metod CPM, HNF in WL (tabela 12).

2.3.4 Primerjava rezultatov

Na osnovi povedanega sledi, da algoritma MinGlDol in MinGlGor dodeljujeta *GPP* ob manjšem (kvečjemu enakem) številu globalnih povezav. Analiza FFT, DAS in LU algoritmov je potrdila našo domnevo, da je število globalnih povezav v tesni zvezi z upadom idealne pospešitve. S tem je upravičena naša odločitev, da smo se pri oblikovanju algoritmov za dodeljevanje osredotočili na minimizacijo števila globalnih povezav. To minimizacijo opravljata algoritma MinGlDol in MinGlGor, ki za svoje delo potrebujeta sinhroniziran *GPP*. Slednjega pa konstruiramo z enim od naših algoritmov TOptSinh oz. pOptSinh. S takšno minimizacijo je dosežen zastavljeni cilj, tj. časovna optimizacija asinhronega procesiranja.

Algoritma MinGlDol in MinGlGor sta bila preizkušena nad 500 *GPP*, pri čemer smo spreminjali: število točk $n \leq 120$, čas njihovega izvrševanja

t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	112	1.68	0.56	0.29
20	152	1.24	0.41	0.75
30	192	0.98	0.33	1.21

Tabela 12: LU: MinGlDol in MinGlGor dodeljevanje pri različnih t_c.

t_c	MinGlGor ($D_p/D_p^{\uparrow}$)	MinGlDol ($D_p/D_p^{\downarrow}$)
5	1.3729	1.3623
10	1.2216	1.2199
20	1.1260	1.1187

Tabela 13: Relativna kvaliteta MinGlDol in MinGlGor algoritmov.

$t(v) \leq 10$ in medprocesorsko komunikacijo $t_c \leq 20$. Število procesorjev smo omejili na $p = 1/2p_{T_\infty}$. Kvaliteto algoritmov MinGlDol in MinGlGor smo ocenjevali[12] glede na algoritem NakljDod. Rezultati so zbrani v tabeli 13, kjer so D_p, $D_p^{\uparrow}$ in $D_p^{\downarrow}$ upadi idealne pospešitve pri algoritmih NakljDod, MinGlGor in MinGlDol.

3. Organizacija stroja

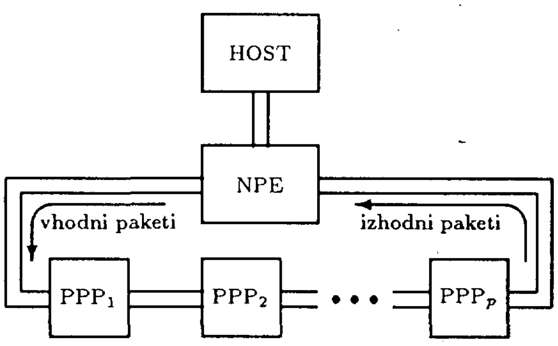
Končno bomo predstavili še računalniško arhitekturo, ki učinkovito podpira podatkovno vodeno računanje, oplemeniteno s sinhronizacijskimi mehanizmi, kot so bili vpeljani v prejšnjem poglavju. Takšna arhitektura v popolnosti izkorišča rezultate časovne optimizacije asinhronega procesiranja.

3.1 Funkcionalni opis

Že uvodoma smo opozorili na problem kopičenja podatkovnih paketov v vrsti pred procesno enoto, ki je posledica omejenega števila procesorjev v procesni enoti. Nakazali smo tudi eno od možnih rešitev, ki temelji na pravilnem vstopanju paketov v vrsto oz. izstopanju paketov iz nje. Na osnovi vpeljanih sinhronizacijskih mehanizmov je to rešitev možno uresničiti z vzporedno računalniško arhitekturo, kot bo opisana v nadaljevanju; poimenovali jo bomo *sinhronizirana podatkovno pretokovna arhitektura* (SPPA) [7, 16]. SPPA sodi v družino hibridnih arhitektur. Ses-

[12] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

tavlja jo množica enakih *podatkovno pretokovnih procesorjev* (PPP) ter *nadzorno-povezovalna enota* (NPE), ki so krožno povezani, kot kaže slika 14. Vhodno/izhodna komunikacija z gostiteljskim računalnikom poteka preko NPE.



Slika 14: SPPA.

Preden se lotimo podrobnejšega opisa enot, na kratko opišimo potek reševanja danega problema na SPPA. Reševanje si lahko predstavljamo kot zaporedje treh faz: prevajanje programa ter optimizacija GPP, nalaganje $SGPP$ v SPPA in izvrševanje v SPPA.

Prevajanje ter optimizacija. Naj bo dan nek poljuben algoritem, zapisan v kakem od visokih podatkovno pretokovnih jezikov. Z ustreznim prevajalnikom se algoritem prevede v strojni kod, tj. GPP. Prevajanje poteka na gostiteljskem računalniku. V naslednjem koraku GPP optimiziramo, tj. konstruiramo p- ali T-optimalno sprožitveno funkcijo (`pOptSinh` in `TOptSinh`) ter z uporabo algoritmov `MinGlGor` oz. `MinGlDol` določimo procesorske indekse π. Ti indeksi so pomemebni za samo nalaganje GPP v SPPA. Poleg tega pa so ob koncu optimizacije znane še vse globalne povezave ter trenutki sprožitve vseh tistih točk (*vhodne ločene* točke), v katere vstopajo globalne povezave. Ti podatki imajo pomembno vlogo pri izvrševanju GPP v SPPA.

Nalaganje. Kot bomo videli v nadaljevanju, ima vsak PPP svoj grafni pomnilnik, v katerem hrani podatke o delu GPP. V fazi nalaganja se vsaka točka v iz GPP naloži v PPP z indeksom i, če je $\pi(v) = i$. V PPP s tem indeksom se implicitno vpišejo tudi vse povezave, katerih začetna in končna točka sta dodeljeni procesorju z indeksom i, kot narekuje logična zgradba grafnega pomnilnika. V NPE pa se shranijo globalne povezave ter trenutki sprožitve vseh vhodnih ločenih točk. Pri nalaganju sodelujeta gostiteljski računalnik ter NPE.

Izvrševanje. Po fazi nalaganja vsak PPP hrani ustrezen podgraf od GPP. Vse točke v PPP, ki za svojo izvršitev potrebujejo vsaj en podatkovni paket iz drugega podgrafa, se imenujejo vhodne ločene točke. Podobno pa so *izhodne ločene* točke vse tiste točke podgrafa, ki vsaj en svoj rezultat posredujejo v drug podgraf. Vse ostale točke v podgrafu imenujemo *notranje* točke. Izmenjava podatkov med notranjimi točkami poteka asinhrono, zato je podgraf *čisti* graf pretoka podatkov brez vnaprej določenih trenutkov sprožitve. Tudi izhodna ločena točka posreduje svoj rezultat takoj, ko se le ta izračuna.

Izmenjava podatkovnih paketov med PPP poteka posredno preko NPE. Vsaka izhodna ločena točka torej pošlje svoj paket v NPE. Ta enota prebere iz seznama globalnih povezav, kateri vhodni ločeni točki je paket namenjen. Poleg tega pa iz seznama sprožitvenih trenutkov ugotovi, kdaj mora ta paket poslati ustreznemu PPP. Vidimo, da NPE deluje popolnoma sinhrono na osnovi globalne ure.

3.2 Nadzorno-povezovalna enota

NPE opravlja naslednje funkcije: skrbi za vhodno/izhodno komunikacijo z gostiteljskim računalnikom, omogoča nalaganje delov GPP v ustrezne PPP in skrbi za posredovanje paketov med ločenimi točkami. Logično si NPE predstavljamo kot tabelo

Globalna povezava	Procesorski indeks	Podatkovno polje	Trenutek sprožitve
$u \longmapsto v$	$\pi(v)$	p	$s(v)$

kjer je $u \longmapsto v$ globalna povezava iz točke u v v, $\pi(v)$ indeks PPP, v katerem je vhodna ločena točka v, p podatkovno polje, ki ga točka u pošilja točki v, ter $s(v)$ trenutek sprožitve točke v. Podatkovno polje p je par $p = (d, a)$, kjer je d vrednost, a pa določa, kateri od argumentov točke v bo sprejel vrednost d.

Tabela se pred začetkom izvrševanja uredi po naraščajočih vrednostih $s(v)$, ki so bile določene v fazi optimizacije. Delovanje NPE opišimo na primeru. Naj bo $A \longmapsto B$ globalna povezava. Točka A naj bo dodeljena procesorju i, točka B pa procesorju j. Točka B se mora sprožiti v trenutku t. V nekem trenutku $\tau < t$ točka A pošlje podatkovno polje p v izhodnem paketu oblike

NPE	$A \longmapsto B$	p

v NPE. Ta na osnovi globalne povezave $A \longmapsto B$ poišče v tabeli vrstico s to globalno povezavo ter v podatkovno polje vpiše vrednost p. S tem je sprejem izhodnega paketa končan.

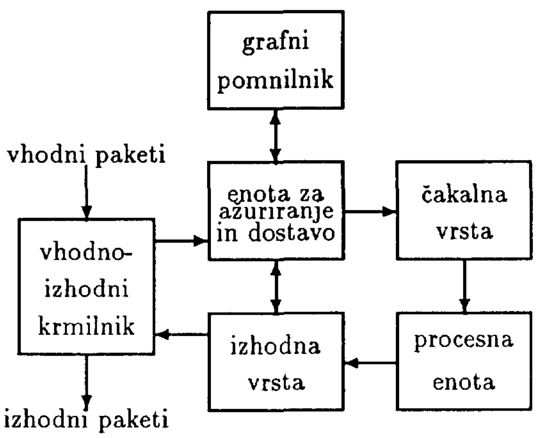
Sočasno globalna ura enote NPE šteje čas τ. Ko postane $\tau = t$, NPE tvori vhodni paket oblike

j	B	p

in ga pošlje verigi PPP, kjer ga j-ti PPP sprejme.

3.3 Podatkovno pretokovni procesor

Podatkovno pretokovni procesor omogoča hranjenje in izvajanje podgrafov GPP ter hkrati učinkovito komuniciranje s svojo okolico. Arhitektura je zasnovana tako, da je uporabljen sinhronizacijski mehanizem, ki temelji na shranjevanju paketov. Procesor sestavljajo *vhodno-izhodni krmilnik, grafni pomnilnik, enota za ažuriranje in dostavo, čakalna vrsta, procesna enota* ter *izhodna vrsta* (slika 15).



Slika 15: Zgradba PPP.

Vhodno-izhodni krmilnik skrbi za komunicajo med PPP in okolico. Vhodni podatkovni paketi, ki vstopajo v krmilnik, imajo obliko

$\pi(v)$	v	p

kjer je v točka, ki (v procesorju z indeksom $\pi(v)$) čaka na podatkovno polje $p = (d, a)$. Iz indeksa $\pi(v)$ krmilnik ugotovi, če je prispeli paket namenjen njemu. Če ni, ga vhodno-izhodni krmilnik nespremenjenega v istem ciklu posreduje naslednjemu PPP. Procesor je tako za tuje pakete transparenten. Če pa se $\pi(v)$ ujema z indeksom danega procesorja, se paket sprejme. V tem primeru enota za ažuriranje in dostavo iz paketa izloči indeks $\pi(v)$, preostanek

v	d	a

pa pošlje grafnemu pomnilniku.

Grafni pomnilnik hrani opise točk in povezav podgrafa GPP. Opis točke v se nahaja v tabeli

Točka	Op	Št.	Operandi			Točke čakajoče na rezultat		
v	op	c	d_1	$\ldots$	d_m	w_1, f_1, a_1	$\ldots$	w_n, f_n, a_n

kjer je *op* operacija, ki jo mora točka v izvršiti nad operandi $d_1, d_2, \ldots, d_m$ ter rezultat poslati točkam $w_1, w_2, \ldots, w_n$; rezultat mora biti shranjen v točko w_i kot argument z zaporedno številko a_i. Zastavica f_i zavzame vrednost L ali G glede na to, ali je točka w_i v istem (lokana točka, $f = L$) ali drugem PPP (globalna točka, $f = G$). Števec c manjkajočih vhodnih operandov se ob nalaganju postavi na vrednost m.

Enota za ažuriranje in dostavo vpiše vrednost d paketa v grafni pomnilnik v vrstico z oznako točke v na mesto operanda d_a ter hkrati zmanjša števec c za eno. Takoj po vpisu enota za ažuriranje in dostavo preveri, če je vrednost števca c enaka nič. Če je temu tako, posreduje paket oblike

v	op	d_1	$\ldots$	d_m

v čakalno vrsto, kjer se paketi kopičijo in čakajo na sprostitev procesne enote. Sočasno pa na osnovi v-te vrstice grafnega pomnilnika tvori v izhodni vrsti naslednjo tabelo

Povezava	Zastavica	Rezultat	Št. argumenta
$v \longmapsto w_1$	f_1		a_1
$v \longmapsto w_2$	f_2		a_2
$\vdots$			
$v \longmapsto w_n$	f_n		a_n

Ko procesna enota izračuna rezultat $r = op(d_1, \dots, d_m)$, tvori paket

v	r

in ga pošlje izhodni vrsti. Ta vpiše vrednost r v polja za rezultat vseh vrstic, ki ta rezultat pričakujejo.

Enota za ažuriranje in dostavo na osnovi zastavice f_i tvori lokalni paket, če je $f_i = L$ oz. izhodni paket, če je $f_i = G$. Lokalni paket ima obliko

w_i	r	a_i

izhodni paket pa obliko

NPE	$v \longmapsto w_i$	d

kjer je $d = (r, a_i)$.

Vrednost r lokalnega paketa se preko enote za ažuriranje in dostavo vpiše v grafni pomnilnik v vrstico z oznako w_i na mesto operanda d_{a_i}. Izhodni paket pa se preko vhodno-izhodnega krmilnika posreduje NPE.

3.4 Izvedba

Zadnji korak do končne izvedbe SPPA predstavlja razvoj strojne opreme NPE in PPP.

V začetni fazi je moč NPE relativno preprosto emulirati s kakim od obstoječih mikroračunalnikov; končni cilj pa je izvedba NPE v obliki VLSI čipa.

Tudi PPP je smiselno zasnovati kot VLSI vezje, vendar pa že danes obstajajo na tržišču procesorji, ki bi v prvi fazi zadovoljivo opravljali funkcijo PPP. Najbliže temu je podatkovno vodeni mikroprocesor μPD7281 [17, 18]. Primeren kandidat, ki bi v prvi fazi prevzel funkcijo PPP, je transputer (npr. T9000) [19].

Za konec omenimo še možnost, da bi PPP nadomestili tudi s podatkovno vodenim procesorskim poljem. Še posebej so zanimiva heksagonalna procesorska polja, ki so trenutno še predmet raziskav [20]. Zgradbo procesorskega polja prikazuje slika 16. Podatkovno vodene *celice* (procesorji) so organizirane v vrstice. Med vsakim parom vrstic je speljano *komunikacijsko vodilo*, ki omogoča vhodno/izhodno komunikacijo z NPE ter porazdelitev delov GPP med celice. Vsaka celica je točkovno povezana s šestimi sosednjimi celicami. Slika 16a prikazuje povezanost celice 0 s sosedami 1, 2, 3, 4, 5 in 6. Iz tehnoloških razlogov je število celic v polju omejeno (trenutno do nekaj 100 [20]). Zato moramo v splošnem GPP porazdeliti po več poljih. Na primer, slika 16b prikazuje rezultat nalaganja tistega dela GPP za FFT, ki je ga je algoritem `MinGlDol` priredil procesorju P_2.

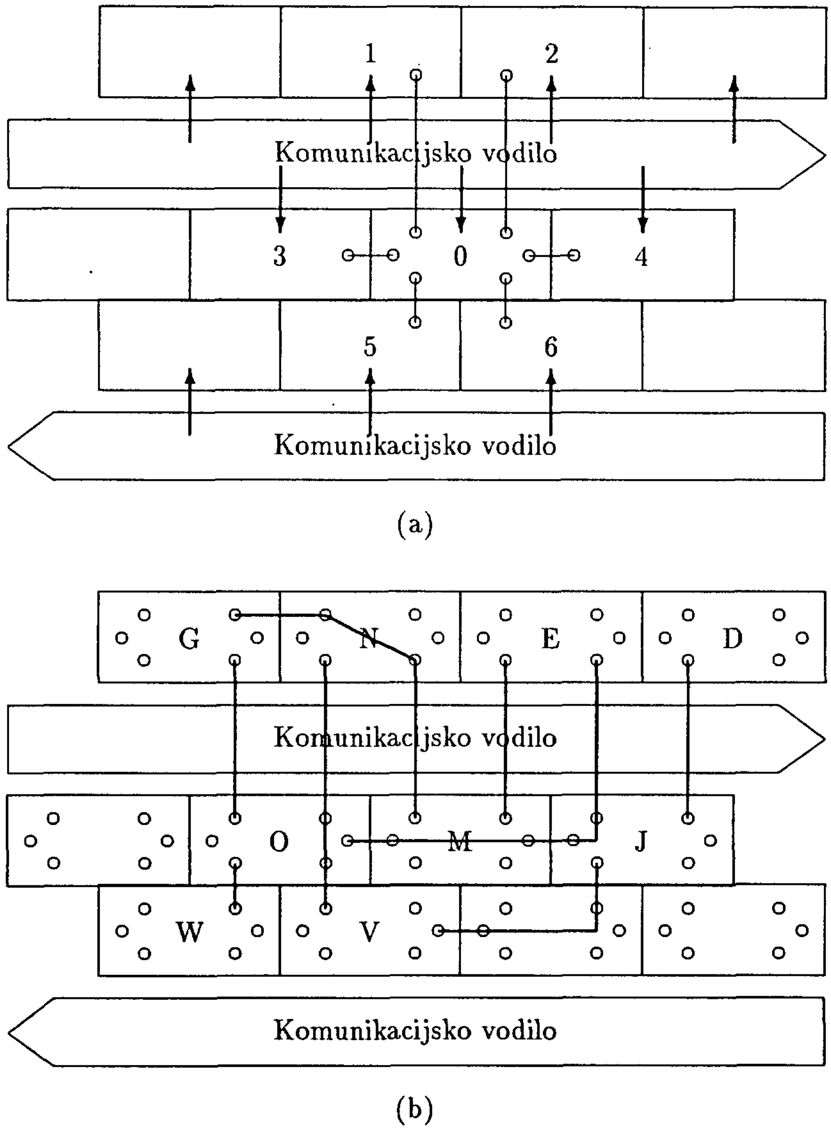
4. Zaključek

Povrnimo se ponovno na nalogo, ki smo si jo zastavili. Predpostavimo, da je na voljo podatkovno pretokovni računalnik s potencialno neskončnim številom procesorjev ter izberimo poljuben graf pretoka podatkov. Denimo, da je za najhitrejšo asinhrono izvršitev izbranega grafa potrebnih vsaj m procesorjev. Tedaj označimo s T_m najkrajši čas, v katerem se ta graf izvrši na m procesorjih. Na računalniku z $n < m$ procesorjev pa bi se v splošnem isti graf asinhrono izvršil v času $T_n = T_m + \Delta T_n$, kjer $\Delta T_n \geq 0$. Za nalogo si zadajmo minimizirati podaljšek izvrševanja ΔT_n, tj. časovno optimizirati asihrono procesiranje pri n danih procesorjih.

Časovno optimizacija asinhronega procesiranja se rešuje bodisi dinamično, tj. med izvrševanjem grafa pretoka podatkov, ali pa statično, tj. med prevajanjem programa v graf. Dinamično dodeljevanje je obremenjeno z "overheadom" ter zaradi lokalne optimizacije redko vodi v globalno (časovno in/ali prostorsko) optimalno izvrševanje grafa pretoka podatkov. Zato je smiselno čim večji del optimizacije prenesti v fazo prevajanja. K razrešitvi opisanega problema smo pristopili po izvirni poti. Rešitev, ki jo predlagamo, temelji na uvedbi nekaterih mehanizmov sinhronizacije v asinhrono procesiranje.

Graf pretoka podatkov s predhodno analizo opremimo z dodatno informacijo, ki bo koristila pri vlaganju grafa v računalnik ter med njegovim izvrševanjem. Postopek, imenovan statično dodeljevanje, poteka v dveh korakih:

Sinhronizacija: Naj bo $\mathcal{V}$ množica točk danega grafa pretoka podatkov ter označimo s $t(v)$ čas



Slika 16: Heksagonalno procesorsko polje.

izvrševanja točke $v \in \mathcal{V}$. Vsaki točki $v \in \mathcal{V}$ hevristično priredimo trenutek njene sprožitve $s(v)$ tako, da minimiziramo ΔT_n, kjer je $n < m$. K hevrističnim metodam se zatečemo zaradi NP-polnosti opisanega problema. Graf, katerega točke so opremljene s trenutki sprožitve, imenujemo sinhronizirani graf pretoka podatkov. Sinhronizirani graf je torej oplemeniten graf, v katerem je znano, kdaj oziroma v kakšnem vrstnem redu se morajo točkam pridruženi ukazi pričeti izvrševati, da se bo celoten graf izvršil na $n < m$ procesorjih v najkrajšem času.

Dodeljevanje: Vsaki $v \in \mathcal{V}$ priredimo indeks $\pi(v)$, $1 \leq \pi(v) \leq n$, ki pove, kateremu od n procesorjev se bo točka dodelila. Torej razbijemo $\mathcal{V}$ v n particij $\mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_n$, kjer je $\mathcal{V}_i$ množica točk, ki se bodo izvrševale na i-tem procesorju. Točki imenujemo ločeni, če sta dodeljeni različnima procesorjema. Povezavo med ločenima točkama imenujemo globalna povezava. Množico $\mathcal{V}$ je v splošnem moč razbiti na več načinov, seveda pa se omejimo le na konstrukcijo dopustnih particij, ki omogočajo izvršitev grafa v skladu z določenimi trenutki sprožitve $s(v)$. Med dopustnimi particijami iščemo optimalno, tj. takšno, pri kateri bo najmanj globalnih povezav. Tako bo zagotovljena tudi najmanjša medprocesorska komunikacija. Rezultat dodeljevanja je, da imamo za vsako $v \in \mathcal{V}$ določen indeks $\pi(v)$ ter "označene" globalne povezave.

Sinhronizirani graf pretoka podatkov dobimo s pomočjo hevrističnih algoritmov pOptSinh in TOptSinh, za konstrukcijo optimalnih sprožitvenih funkcij. Po pesimistični oceni vračata predlagana algoritma optimalno rešitev v 80% primerov.

V algoritmu pOptSinh je uporabljena izvirna ocena za spodnjo mejo potrebnih procesorjev, ki temelji na razširjeni kritični vzporednosti grafa. Ta ocena je po svoji natančnosti v povprečju le za 6.52% slabša od optimalne Fernandez-Bussellove ocene, hkrati pa je njeno določanje za red velikosti hitrejše.

Za dodeljevanje predlagamo algoritma MinGlDol in MinGlGor, ki temeljita na optimizaciji medprocesorskih komunikacij. V primerjavi z znanimi razvrščevalnimi algoritmi dobimo s predlaganima algoritmoma boljše rezultate, kar potrjujejo analizirani primeri algoritmov za izračun hitre Fourierjeve transformacije, dinamične analize scene in LU razcepa matrike.

Končno je predlagana tudi organizacija računalnika, ki podpira asinhrono procesiranje z elementi sinhronizacije. Nakazane so možnosti realizacije z VLSI podatkovno pretokovnimi mikroprocesorji in podatkovno pretokovnimi polji.

Zahvala

Raziskavo je finančno podprlo Ministerstvo za znanost in tehnologijo Republike Slovenije po pogodbi C2-0521-106-92. Za strokovno pomoč pri nastanku pričujočega dela gre zahvala mag. Borutu Robiču, sodelavcu Laboratorija za računalniške arhitekture IJS.

Literatura

[1] E. B. Fernandez and B. Bussell. Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules. *IEEE Trans. Computers*, C-22(8):745–751, August 1973.

[2] Y. E. Chen and D. L. Epley. Bounds on Memory Requirements of Multiprocessing Systems. In *Proc. 6th Annu. Allerton Conf. Circut and Syst. Theory*, pages 523–531, 1968.

[3] R. McNaughton. Scheduling with Deadlines and Loss Functions. *Management Sci.*, 6, October 1959.

[4] T. C. Hu. Parallel Sequencing and Assembly Line Problems. *Oper. Res.*, 9(6):841–848, November 1961.

[5] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez. Optimal Schedulling Strategies in a Multiprocessor System. *IEEE Trans. Computers*, C-21(2):137–146, February 1972.

[6] J. Šilc, B. Robič, and L. M. Patnaik. Performance Evaluation of an Extended Static Dataflow Architecture. *Computers and Artificial Intelligence*, 9(1):43–60, 1990.

[7] J. Šilc and B. Robič. Synchronous Dataflow-Based Architecture. *Microprocessing and Microprogramming*, 27(1-5):315–322, August 1989.

[8] J. Šilc and B. Robič. Program Graph Partitioning for Macro-Dataflow. In *Proc. ISSM Int'l Workshop on Parallel Computing*, pages 327–330, September 1991.

[9] J. Šilc. *Časovna optimizacija asihronega procesiranja z uvedbo nekaterih mehanizmov sinhronizacije.* Doktorska disertacija, Fakulteta za elektrotehniko in računalništvo, Ljubljana, junij 1992.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[11] K. Mehlhorn. *Graph Algorithms and NP-Completeness.* Springer-Verlag, 1984.

[12] E. G. Coffman et al. *Computer and Job-Shop Scheduling Theory.* Wiley-Interscience, 1976.

[13] B. Shirazi, M. Wang, and G. Pathak. Analysis and Evaluation of Heuristic Method for Static Task Schedulling. *Journal of Parallel and Distributed Computing*, 10(3):222–232, November 1990.

[14] G. Pathak and D. P. Agrawal. Task Division and Multicomputer System. In *Proc. 5th Int'l Conf. on Distributed Computing System*, page 273, May 1985.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

[16] J. Šilc and B. Robič. MADAME - Macro-Dataflow Machine. In *Proc. Mediterranean Electrotechnical Conference - MELECON'91*, pages 985–988, May 1991.

[17] T. Jeffery. The μPD7281 Processor. *Byte*, pages 237–246, November 1985.

[18] J. Šilc in B. Robič. Procesor s podatkovno pretokovno arhitekturo. *Informatica*, 10(4):74–80, 1986.

[19] D. Pountain. The Transputer Strikes Back. *Byte*, 16:265–275, August 1991.

[20] S. Weiss, I. Spillinger, and G. M. Silberman. Architectural Improvements for Data-Driven VLSI Processing Arrays. In *Proc. Functional Programming Languages and Computer Architecture*, pages 243–259, September 1989.