Programok leképezése különbözo architektúrákra, programsémák

A különbségeket a folyamatok kommunikációi jelentik elsősorban.

Architektúrák

Aszinkron, megosztott memóriaarchitektúra

Az M memória közös, a "rekeszeit" sorszámaikkal érjük el. Adottak a p_i processzorok, mindnek külön c_i órajele van. A program legyen $S = (s_0, \{s_1, \dots s_m\})$. Minden S-beli s_i utasításhoz rendeljünk egy processzort, ami őt futtatja (nyilván egy processzor többet is futtathat): $s_i \to p_j$. A processzor ezeket ismételgeti naphosszat.

Ha v_k s_i baloldali változója (azaz aki értéket kap a folyamatban), (azaz ha $v_k \in LV(s_i)$) akkor az s_i-hez rendelt p_j olyan, hogy joga van írni azt az m_z memóriacellát, ahol v_k van.És a jobboldali változókra pedig hasonlóan csak olvasásra.

Osztott rendszer

Itt szintén minden processzorhoz tartozik külön órajel, na de még külön memória is. Egymáséiba nem látnak bele. A kommunikáció itt csatornaváltozókon keresztül valósul meg. Mindenkinek vannak tehát $v \in V$ saját változói és akad néhány $c \in Ch$ csatorna. Esetleg a csatornákhoz van puffer (sor). Két processzor között fizikailag egy csatorna lehet maximum.

Szinkron, megosztott rendszer

Egy, közös órajele van a processzoroknak, a memória is közös. Írhatnak ketten ugyanoda, de csak ugyanazt. Olvasni szabad többnek is egyszerre. Hosszú, vektorszerű értékadások esetén használatos, mely sok-sok változót érint.

Programsémák

Read-only

Egy program minden utasítása egy processzorhoz tartozik (ugyanakkor egy processzorhoz több utasítás is tartozhat). Egy ilyen program akkor van a read-only sémában, ha minden változójára igaz, hogy legfeljebb egy processzorhoz tartozó utasítások baloldalán fordul elő. (max egy processzor írhasssa)

- finom szemcsézettség legfeljebb egy idegen változó legfeljebb egyszer van utasítás jobboldalán (baloldalon nem is lehet, mert readonly, mindenkit csak egy processzor írhat, a többi az "idegen") Példa:

$$s : x := x + y + z \parallel y := y + 1$$
$$s' : z := \dots$$

Itt egy darab idegen változó van (a z). Ha pl s'-ben y is értéket kap, akkor az már nem jó.

- durva szemcsézettség Továbbra is igaz, hogy minden változót legfeljebb egy processzor ír, de most egynél több idegen változó is megengedhető. "Elkérni" nem lehet változót, mert akkor holtpont lesz a blokkolások miatt.

$$s : x := x + y + z \parallel y := y + 1$$
$$s' : z := \dots$$
$$s'' : y := y + 1$$

$\sim$ aszinkron megosztott rendszer architektúra

Közös változó séma

A v változó lokális p processzorra, ha csak a p-hez rendelt értékadásokban fordul elő (tehát máshol még olvasásra se). Akkor van ebben a sémában, ha minden értékadásra teljesül, hogy legfeljebb egy nem lokális változó van benne (akár baloldalon, akár jobboldalon... összesen) (max egy processzor nyúlhat hozzá) Ha a fenti két séma kikötései nem teljesülnek, akkor rögzített sorrendben kell az előkerülő nemlokális/idegen változókat lefoglalni, ha el akarjuk kerülni a holtpontot.
$\sim$ osztott rendszer architektúra

Egyetlen utasítás séma

Szinkron architektúrán jól kezelhető. A program így néz ki: $S = (s_0, \{\dots, \parallel_i v_i := \dots, \dots\})$. Tehát mindegyik egy-egy vektorszerű értékadást csinál. Ekkor ha mondjuk ez a programunk: $S = (s_0, \{x := f(x,y) \parallel y := g(x,y)\})$, akkor valamekkora párhuzamosság (aszinkronitás) megengedett (tehát ilyen architektúrán is tud működni (?)), elég csak bizonyos pontokon bevárni egymást pl, amíg nincs meg $f(x,y)$ kiszámítása x_1-re, addig még g se foglalkozzon y_2-vel és viszont. De ezen kívül már házon belül lerendezhetik az ütemezést.

Egyenlőségi séma

Egyenlőségi séma.
funkcionális programozás
definiált fv-ek definícióinak sorrendje (?)
példa rá 7/a tétel
TODO ábrák