VI. Paralelism Implicit- Paralelism la nivel de instructiune: Procesoare pipeline, superscalare si vectoriale.

Continut

- Pipeline
- Procesoare vectoriale
- Procesoare superscalare
- Probleme de programare

Arhitectura traditionala scalara a lui von Neumann

- Ofera un singur fllux de control asupra cu instructiuni executate serial ce opereaza asupra unor operanzi scalari.
- Procesorul are o unitate de executie a instructiunilor (IEU).
- Executia unei instructiuni poate fi pornita numai dupa ce instructiunea anterioara in flux a fost terminata
- Exceptand un numar mic de instructiuni speciale de transfer intre memoria principala si registrii, instructiunile iau operanzii din si pun rezultatele in registrii scalari
  - Un registru scalar este un registru care contine un singur intrega sau un numar in virgula mobila
- Timpul total de executie a programului este egal cu suma timpilor de executie a instructiunilor.
- Performanta acelei arhitecturi este determinata de frecventa ceasului.
- Toate componentele – procesor, memorie, si calea datelor – pot produce incetinirea procesarii
  - Inovatiile in domeniul arhitecturii din ultimii ani adreseaza aceasta probl.
  - Una dintre cele mai importante este multiplicitate – in unitati de procesare, cai catre date si unitati de memorie.

Pipeline (conducta)

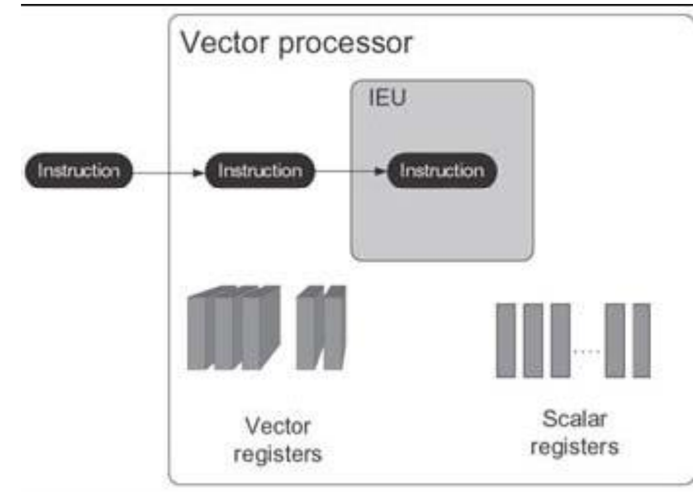
- Suprapunerea stadiilor variate in executia instructiunii (inaintare, planificare, decodare, obtinere operanzi, executie, stocare etc) permite executia mai rapida.
- Analogia cu o linie de asamblare este utila pentru intelegerea pipeline.
  - Daca asamblarea unei masini necesita 100 unitati de timp care pot fi impartite in 10 etape de catre 10 unitati fiecare, linia de asamblare poate produce o masina la fiecare 10 unitati de timp!
    - Reprezinta o accelelare de 10 ori fata de cazul serial.
  - Pentru cresterea vitezei unui pipeline este necesara impartirea sarcinilor in unitati din ce in ce mai mici, lungind pipeline si crescand suprapunerea in executie.
  - In contextul procesoarelor, acest lucru permite frecvente de ceas mai mari pentru ca sarcinile sunt mai mici
- Exemple: Pentium 4 opereaza la 2.0 GHz si are 20 etape pipeline.
- La mașinile moderne, în principal, toate operațiunile sunt conducte:
  - sunt necesare mai multe etape hardware pentru a face orice calcul.
  - Este posibil să se facă simultan mai multe operantii.
- Această noțiune de operații pe conducte nu a fost inventată ieri.
  - Proiectul Universității din Manchester Atlas a implementat astfel de conducte aritmetice încă din 1962.

Pipelining

- Terminologia este o analogie a unei lungimi scurte de țeavă în care începe să împingă marmure:
  - Imaginează-ți că țeava va ține bucati L, care vor fi simbolice pentru etapele necesare procesării fiecărui operand.
  - Pentru a face o operație completă pe o pereche de operand, trebuie să faceți pași L.
  - Cu mai multi operanzi, putem continua să împingem operanzii în conductă până când este plin, după care un rezultat (marmură) apare celălalt capăt cu o viteză de un rezultat / ciclu.
  - Prin acest dispozitiv, în schimb de n operanzi care iau L cicluri fiecare, adică un total de n · L cicluri, sunt necesare doar cicluri L + n, atât timp cât ultimele operanzi pot fi scoase din conductă odată ce au pornit în acesta.
  - Viteza de viteză rezultată este n · L / (n + L), adică L pentru n mare.
- Viteza unei conducte este limitată de cea mai mare sarcină atomică din conductă.
- Statistic, fiecare a cincea până la a șasea instrucțiune este o instrucțiune cu ramificatie.
  - Prin urmare, conductele lungi de instruire au nevoie de tehnici eficiente pentru a prezice destinațiile sucursalei, astfel încât conductele să poată fi completate speculativ.
  - Pedeapsa unei interpretări greșite crește pe măsură ce conductele devin mai profunde, deoarece un număr mai mare de instrucțiuni trebuie inaintate.
  - Acești factori plasează limitări la adâncimea unei conducte de procesare și la câștigurile de performanță rezultate.

Procesor vectorial

- Oferă un singur flux de control cu instrucțiuni executate în serie care operează atât pe operanzi vectoriali cât și scalari.
- Paralelismul acestei arhitecturi este la nivelul instrucțiunilor.
- Ca și procesorul scalar serial:
  - are un singur IEU: nu începe să execute următoarea instrucțiune decât până la executarea celei curente.
- Spre deosebire de procesorul scalar serial:
  - instrucțiunile sale pot funcționa atât pe operanzi scalari cât și pe operanzi vectori.
- Vectorul operand:
  - este un set ordonat de scalare care este în general localizat pe un registru vectorial.



-Implementari ale acestei arhitecturi: ILLIAC-IV, STAR-100, Cyber-205, Fujitsu VP 200, ATC,

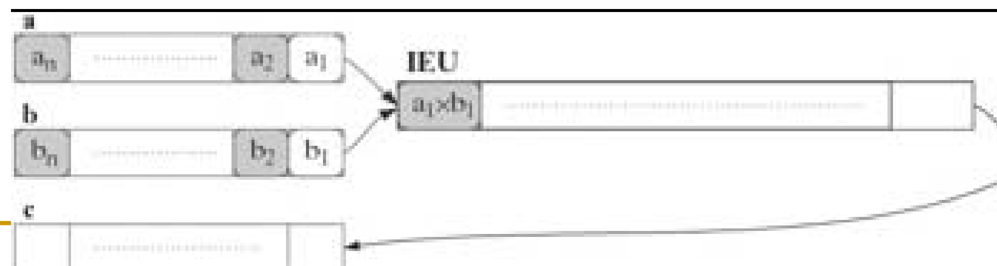
-Cel mai elegant este Cray-1 din 1976.

-utilizează o conductă de date pentru a executa instrucțiunile vectoriale.

Exemplu – procesorul vectorial (1)

- Înmulțirea a doi operanzi vectoriali,
  - Extensii în funcție de element ale operației scalare corespunzătoare.
- Operanzi in registrii vectoriali **a** si **b**, rezultatul fiind in registrul vectorial **c**.
- Instrucțiunea este executată de o unitate de tip pipeline capabilă să înmulțească scalari.
  - Înmulțirea a doi scalari este împărțită în etape *m*, iar unitatea poate efectua simultan diferite etape pentru diferite perechi de elemente scalare ale operanzilor vectoriali.
- Executarea instrucțiunii vectoriale **c** = **a x b**, unde **a**, **b**, si **c** sunt registrii vectoriali cu *n* elemente, printr-o conducta poate fi sumarizat astfel:

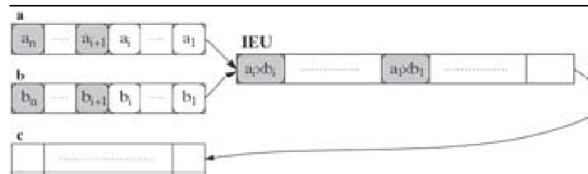
1. In prima etapă, unitatea realizează etapa 1 a înmulțirii elementelor a1 și b1:



Exemplu – procesoul vectorial (2)

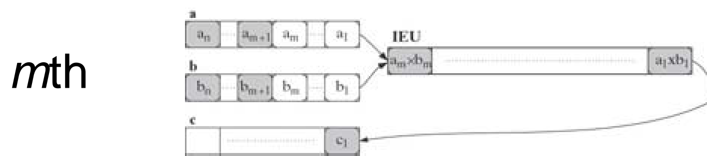
Pasul *i* (*i* = 2,…, *m* - 1), unitatea efectuează în paralel:

- etapa 1 a înmulțirii elementelor ai și bi,
- etapa 2 a înmulțirii elementelor ai-1 și bi-1, …
- stadiul i al înmulțirii elementelor a1 și b1,

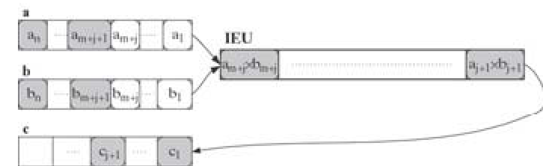


Following steps:

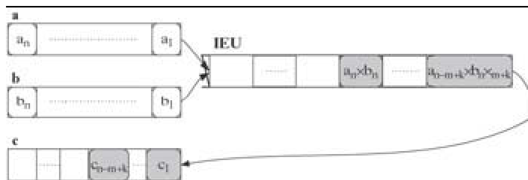
*m*th



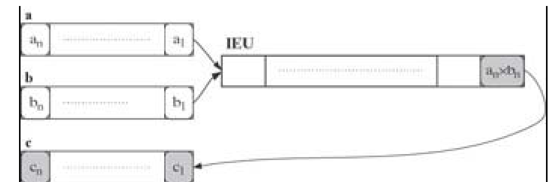
(*m*+*j*)th



(*n*+*k*-*1*)th



(*n*+*m*-*1*)th
(Final)



Exemplu –procesor vectorial (3)

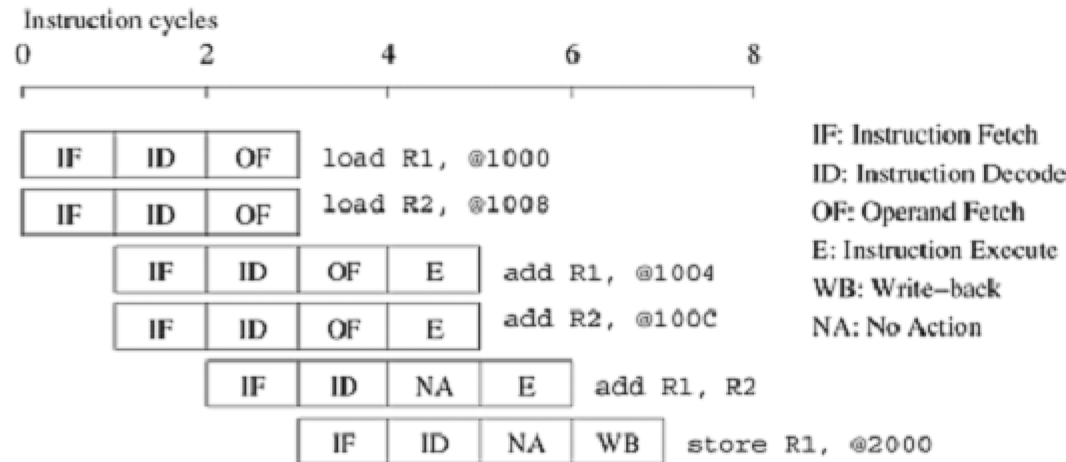
- În total, este nevoie de $n + m - 1$ pași pentru a executa această instrucțiune.
- Conducta unității este complet încărcată numai de la m-a până la a n-a etapă a execuției.
- Executarea în serie a n multiplicări scalare cu aceeași unitate ar lua $n \times m$ pași.
- Acceleratia oferită de instrucțiunea vectorială este $S = n \times m / (n+m-1)$
- Dacă n este suficient de mare, viteza este aproximativ egală cu lungimea conductei unității, $S \approx m$.
- Arhitecturile vectoriale sunt capabile să accelereze astfel de aplicații, o parte semnificativă a cărei calcule se încadrează în operațiunile de bază ale elementelor din tablouri.
- Arhitectura vectorială include arhitectura scalară în serie ca un caz particular ($n = 1$, $m = 1$).

Executie super-scalara

- Un mod evident de a îmbunătăți rata de execuție a instrucțiunilor dincolo de acest nivel este utilizarea mai multor conducte.
  - În timpul fiecărui ciclu de ceas, mai multe instrucțiuni sunt conectate în procesor în paralel.
  - Aceste instrucțiuni sunt executate pe mai multe unități funcționale.
- Capacitatea unui procesor de a emite mai multe instrucțiuni în același ciclu este denumită execuție superscalara.
- Exemplu:
  - Fie un procesor cu două conducte și posibilitatea de a emite simultan două instrucțiuni.
  - Aceste procesoare sunt uneori denumite și procesoare s*uper-pipeline*.
  - Două inaintari pe ciclu de ceas=> it is also referred to as two-way superscalar or dual issue execution.

Exemplu

- adunarea a 4 numere.
- Prima și a doua instrucțiune sunt independente și sunt emise concomitent:
  - load R1, @1000
  - load R2, @1008 at t=0
- Programul presupune că fiecare acces la memorie are un singur ciclu. În realitate, este posibil să nu fie cazul.



(a) Three different code fragments for adding a list of four numbers.

(b) Execution schedule for code fragment (i) above.

Probleme: dependența dintre instrucțiuni

- Instrucțiunile dintr-un program pot fi legate între ele.
- Dependenta datelor:
  - Rezultatele unei instrucțiuni pot fi necesare pentru instrucțiunile ulterioare.
  - Dependențele trebuie rezolvate înainte de emiterea simultană a instrucțiunilor.
  - ⇒ întrucât rezoluția se face în timpul rulării, aceasta trebuie să fie acceptată de hardware.
  - ⇒ cantitatea de paralelism la nivel de instructiune într-un program este adesea limitată și este o funcție a tehnicii de codare.
  - în multe cazuri este posibil să extragem mai mult paralelism prin reordonarea instrucțiunilor și prin modificarea codului.
- Dependența de resurse:
  - resurse finite partajate de diverse conducte.
  - Exemplu:
    - co-programarea a două operații cu punct flotant pe o mașină cu două emisiuni cu o singură unitate in virgule mobila.
    - Instrucțiunile nu pot fi programate împreună, deoarece ambele au nevoie de unitatea in virgule mobila.
- Dependența de ramură sau dependența procedurală:
  - Întrucât destinația ramurii este cunoscută numai la punctul de execuție, instrucțiunile de planificare a priori pe ramuri pot duce la erori.

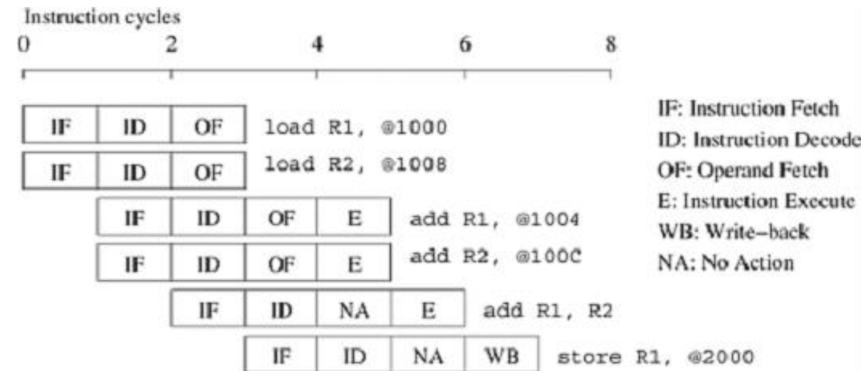
Capabilități pentru microprocesoare

- Capacitatea unui procesor de a detecta și programa instrucțiuni concomitente este esențială pentru performanța suprascalară.
- De exemplu:
  - detectați că este posibil să programați a treia instrucțiune - load R2, @ 1008 - cu prima instrucțiune.
- Majoritatea microprocesoarelor actuale sunt capabile să emită și să finalizeze ordinal
  - Modelul, denumit ordin de instrucțiune dinamică, exploatează maximal paralelismul la nivelului de instrucțiune
  - Procesorul folosește o fereastră cu instrucțiuni din care selectează instrucțiuni pentru emitere simultană
- Microprocesoarele actuale acceptă, de obicei, până la patru ordine de execuție superscalara.
- Datorita:
  - paralelismului limitat,
  - dependenta resurselor, sau
  - incapacitatea unui procesor de a extrage paralelismul,

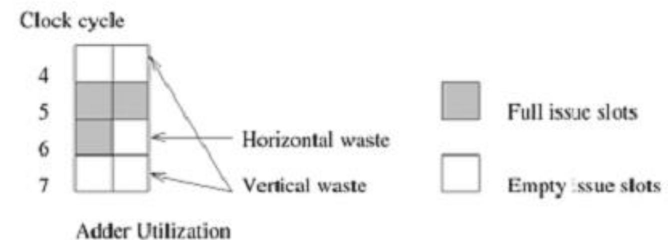
  resursele procesoarelor superscalare sunt foarte slab utilizate.
- Paralelismul extras de procesoarele superscalare este limitat de instrucțiunile de inaintare.
  - Logica hardware pentru analiza dependenței dinamice se situează, de obicei, în intervalul 5-10% din logica totală a microprocesoarelor convenționale
    - aproximativ 5% pe superscalarul cu patru căi Sun UltraSPARC.
  - Complexitatea crește aproximativ în mod cvadratic cu numărul de ordine si poate deveni un blocaj.

Performanța arhitecturii supracalare limitată de paralelismul nivelului de instrucțiune disponibil

- In (c) 2 unitati de executie (multiply-add units),
- Ilustrează mai multe cicluri cu zero ordine:
  - Cicluri în care unitatea in virgule mobile este inactiva.
  - Acestea sunt în mod esențial cicluri pierdute din punctul de vedere al unității de execuție.
- Dacă în timpul unui anumit ciclu nu sunt emise instrucțiuni cu privire la unitățile de execuție, acestea sunt denumite deșeuri verticale.
- Dacă se folosește doar o parte din unitățile de execuție în timpul unui ciclu, aceasta fenomen este asociat cu "deșeuri orizontale".
- In exemplu, avem două cicluri de deșeuri verticale și un ciclu cu deșeuri orizontale.
  - În total, numai 3 din cele 8 cicluri disponibile sunt utilizate pentru calcul.
  - ⇒ fragmentul de cod nu va produce cel mult trei optimi din numărul maxim de FLOP al procesorului.



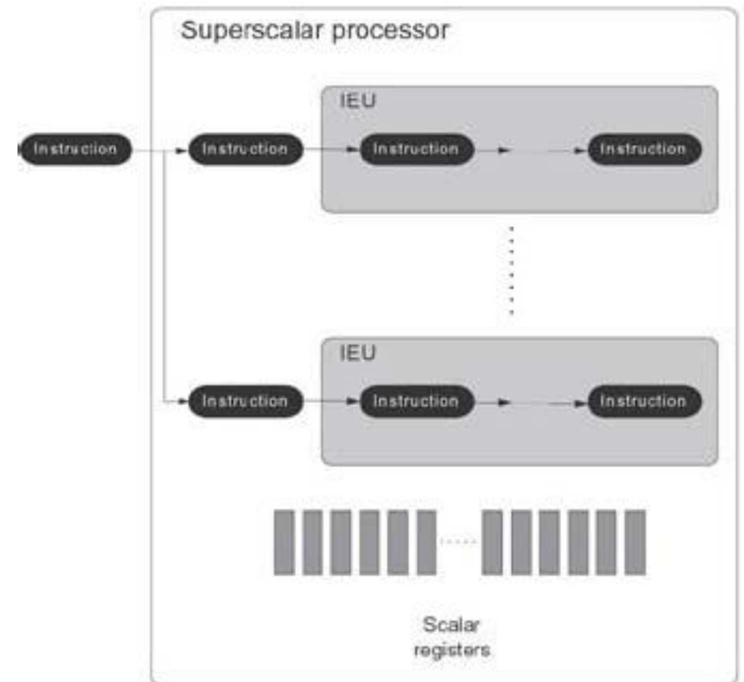
(b) Execution schedule for code fragment (i) above.



(c) Hardware utilization trace for schedule in (b).

Procesor superscalar și operanzi scalari

- Procesorul superscalar asigură un singur flux de control, cu instrucțiuni care operează pe operanzi scalari și care sunt executate în paralel.
- Procesorul superscalar are mai multe unități de execuție a instrucțiunilor care execută instrucțiuni în paralel.
- Cu excepția unui număr mic de instrucțiuni speciale pentru transferul de date între memoria principală și registre, instrucțiunile funcționează pe operanzi scalari aflați pe registrele scalare.
- CDC 6600 (1964) a fost primul procesor cu mai multe unități IEU care funcționează în paralel.
- CDC 7600 (1969) a fost primul procesor cu mai multe unități UE pipeline funcționând în paralel.
- În zilele noastre, practic toate microprocesoarele fabricate au multe unități IEU, fiecare fiind în mod normal o unitate pipelinată și cu greu ne putem imagina alte arhitecturi de microprocesoare.



Două instrucțiuni succesive pot fi executate în paralel de două UI-uri diferite dacă nu au operanzi în conflict, adică,

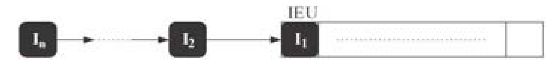
- dacă nu scriu în același registru
- nici o instrucțiune nu folosește un registru în care celălalt scrie.

Superscalar și conducte

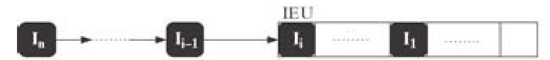
- O unitate de dispecerat care direcționează instrucțiunile către IEU-uri relevante în funcție de tipul lor.
- Fiecare IEU este caracterizat prin setul de instrucțiuni pe care le execută.
- Întregul set de instrucțiuni este împărțit într-un număr de sub-seturi care nu se intersecteaza, asociate fiecărui IEU.
- În plus, fiecare IEU poate fi o unitate pipeline:
  - poate executa simultan mai multe instrucțiuni succesive, fiecare aflându-se pe stadiul de execuție.
- Exemplu:
  - o unitate pipelinată care execută o serie de instrucțiuni independente succesive.
  - Fie conducta unității să fie compusă din m etape.
  - Fie ca n instrucțiuni succesive I1,…, In, să fie efectuate de unitate.
  - Instrucțiunea Ik preia operanțele din registrele ak, bk și pune rezultatul pe registrul ck (k = 1,…, n).
  - Să nu existe două instrucțiuni care să ofere operanzi conflictuali.

Exemplu

- La primul pas, efectuează etapa 1 a instrucțiunii I1:

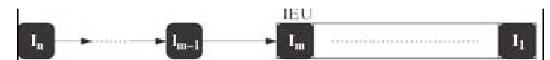


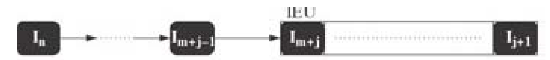
- La pasul ith (i = 2,…, m - 1), unitatea îndeplinește în paralel etapa 1 al instrucțiunii Ii, etapa 2 al instrucțiunii Ii-1 etc., și etapa i al instrucțiunii I1,



...

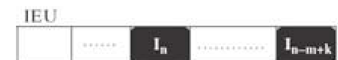
- La (n + m - 1) a treia etapă, unitatea efectuează numai etapa finală m a instrucțiunii In; după finalizarea acestui lucru, registrul cn conține rezultatul instrucțiunii In





Este nevoie de (n + m-1) pași pentru executie.

Conducta unității este complet încărcată numai de la a m-a până la a n-a etapă a execuției.



Executarea stricta în serie de către unitatea de n instrucțiuni succesive durează n x m pași.

Acceleratia maximă oferită de această unitate pipelinată este S = n x m / (n + m-1).

Dacă n este suficient de mare, acceleratia *S* ≈ m.

Very Long Instruction Word Processors

- Un concept alternativ pentru exploatarea paralelismului la nivel de instrucțiuni.
- Se bazează pe compilator pentru a rezolva dependențele și disponibilitatea resurselor la timp de compilare.
  - Instrucțiuni care pot fi executate concomitent:
    - sunt ambalate în grupuri,
    - coletat la procesor ca un singur cuvânt lung de instrucțiune.

    pentru a fi executat pe mai multe unități funcționale în același timp.
  - Instrucțiuni paralele suplimentare sunt puse la dispoziția compilatorului pentru a controla execuția paralelă.
- Utilizat pentru prima dată în Multiflow Trace (1984) și, ulterior, ca o variantă în arh. Intel IA64.
- Avantaje:
  - Planificarea se face în software.
  - Mecanismele de decodare și instrucțiuni sunt simple.
  - Compilatorul
    - are un context mai mare din care să selecteze instrucțiunile,
    - poate utiliza o varietate de transformări pentru a optimiza paralelismul.
- Dezavantaje:
  - compilatoarele nu au starea dinamică a programului (de exemplu, bufferul istoricului ramificarilor) disponibil pentru a lua decizii de planificare.
  - Alte situații de rulare, cum ar fi reluarea preluarii datelor din cauza ratărilor de cache sunt extrem de dificil de prezis cu exactitate
  - Performanța este sensibilă la capacitatea compilatorului de a detecta datele și dependențele de resurse și de a citi și scrie și de a programa instrucțiuni pentru paralelismul maxim.
- Limită la scări mai mici de concurență în intervalul de la patru până la opt ori.

Comun: Procesoare vectoriale si super-scalare

- Arhitecturile vectoriale și suprascalare sunt unite într-un grup din mai multe motive:

1. Arhitecturile vectoriale de succes sunt apropiate de arhitecturile superscalare atât ideologic cât și în implementare.
   - o unitate pipeline vectorială este o clonă specializată a unității de tip superscalar cu scop general, care este optimizată pentru execuția pipeline a n instrucțiuni succesive efectuate ca aceeași operație, dar pe operanzi diferiți.
2. Proiectarea unor procesoare superperscalare avansate, cum ar fi Intel i860, este în mod evident influențată de arhitectura vectorială.
3. Partajeaza același model de programare:
   - Un program portabil bun care poate profita din plin de potențialul de performanță al procesorului superscalar este definit în același mod ca și pentru procesorul vectorial.

- Spre deosebire de procesorul vectorial, procesorul superscalar permite amestecurilor mai sofisticate de operații pentru a-și încărca eficient unitățile pipeline decât simpli operanzi matriceali:
  - chiar dacă arhitectura supracalara pare mai bogata decât arhitectura vectoriala, modelul real de programare utilizat pentru procesoare supracalare ar trebui să fie același ca pentru procesoare vectoriale.

Ciclu desfăşurat– un exemplu de programare

Ciclul simplu

```
for(i=0;i<n;i++){ b[i] = f(a[i]);}
```

este extins în segmente de lungime (să zicem) *m i*.

```
for(i=0;i<n;i+=m){
        b[i ] = f(a[i ]);
        b[i+1] = f(a[i+1]);
        …
        b[i+m-1] = f(a[i+m-1]);
        }
/* residual segment res = n mod m */
nms = n/m; res = n%m;
for(i=nms*m;i<nms*m+res;i++){
    b[i] = f(a[i]);
}
```

Prima buclă prelucrează nms segmente, fiecare dintre ele efectuând m operații f(a[i]).

Ultima noastră buclă curăță restul (un segment rezidual).

Uneori, acest segment rezidual este procesat mai întâi, alteori ultimul (așa cum este arătat) sau din motive de aliniere a datelor, o parte din rezultat.

Ne vom referi la instrucțiunile care procesează fiecare f(a[i]) ca **sablon**.

Optimizare: alegeți o adâncime adecvată de derulare m care permite să strângeți toate șabloanele m împreună în cea mai strânsă grupare de timp posibila.

Pre-preluarea ca etapă de desfășurare a ciclului

- Pregătirea datelor din segment care vor fi folosite de elementele de segment ulterioare pentru a ascunde latențele de memorie.
  - pentru a ascunde timpul necesar pentru a obține datele din memorie în registre.
  - astfel de pre-preluare a datelor a fost numită încărcarea de jos în vremurile anterioare.
- Pre-preluarea în forma sa cea mai simplă este pentru m = 1 și ia forma:

```
t = a[0]; /* prefetch a[0] */
for(i=0;i<n-1; ){
        b[i] = f(t);
        t = a[++i]; /* prefetch a[i+1] */
}
b[n-1] = f(t);
```

  unde se încearcă ascunderea următoarei sarcini a [i] sub surplusul ciclului.
- Scopul desfășurării ciclurilor: ascunde latențele, în special întârzierea la citirea datelor din memorie:
  - Cu excepția cazului în care rezultatele stocate vor fi necesare în iterațiile ulterioare ale ciclului (o dependență de date), aceste stocari pot fi întotdeauna ascunse.

Planificarea instrucțiunilor cu decuplarea ciclurilor

- Fie următoarea operație care trebuie efectuată asupra lui **A**: **B** = *f*(**A**), unde *f*(·) se face in doi pasi : $Bi = f2(f1(Ai))$.
- Fiecare etapă a calculului durează ceva timp și vor exista latențe între ele în cazul în care rezultatele nu sunt încă disponibile.
- Dacă încercăm să executăm împreună, $Bi = f(Ai)$ si $Bi+1 = f(Ai+1)$,
  - diferitele operațiuni: preluarea memoriei, f1 și f2, ar putea rula simultan,
  - Se pot configura două șabloane și am încerca să le aliniem.
    - Pornind pașii de operație f (Ai + 1) cu un ciclu după f (Ai), cele două șabloane pot fi îmbinate împreună.
- Fiecare calcul f (Ai) și f (Ai + 1) ia un anumit număr (să zicem m) de cicluri.
  - Dacă aceste două calcule ar rula secvențial, ar lua de două ori ceea ce necesită fiecare, adică 2m.
  - Fuzionând cele două și alinindu-le pentru a completa golurile, acestea pot fi calculate în m + 1 cicluri.
- Aceasta va funcționa numai dacă:
  1. operațiunile separate pot rula independent și concomitent,
  2. dacă este posibil să aliniați șabloanele pentru a completa unele lacune,
  3. sunt destule registre.
     - Dacă există doar opt registre, alinierea a două șabloane este tot ceea ce pare posibil la momentul compilării.

Dependenta datelor

- Fie urmatorul ciclu în care f (x) este o anumită funcție arbitrară,

```
for(i=m;i<n;i++){ x[i]=f(x[i-k]); }
```

  x[i-k] trebuie calculat înainte x[i] unde *k* > 0

- Cotra-exemplu:

```
for(i=m;i<n;i++){x[i]=f(x[i-k]);}
```

  Noi sau compilatorul putem derula ciclul în orice mod dorit.

  Dacă n - m este divizibil cu 2, se poate derula ciclul de mai sus cu o dependență în grupuri de 2,

```
for(i=m;i<n;i+=2){
      x[i ] =f(x[i-k ]);
      x[i+1]=f(x[i-k+1]);}
```

Ramificari si executie conditionata

- Exemplu:

```
for(i=0;i<n;i++){
        if(e(a[i])>0.0){
                c[i]=f(a[i]);
        } else {
                c[i]=g(a[i]);
    } }
```

- Conditiade ramificare *e*(*x*) este de obicei simpla, dar este probabil să dureze câteva cicluri.
- Cel mai adecvata este vectorizarea:
  - fie f și g și unesc rezultatele, sau
  - alternativ, părți ale ambelor.

Ierarhia memoriei

- Procesoarele vectoriale și superscalare au o ierarhie de memorie pe două niveluri:
  1. Memorie registru mică și rapidă.
  2. Memorie principală mare și relativ lentă.
- Această ierarhie a memoriei este reflectată în seturile de instrucțiuni și este direct vizibilă pentru programatori.
- O simplă ierarhie de memorie reală include următoarele niveluri:
  1. Register memory
  2. Cache memory
  3. Main memory
  4. Disk memory
- Situația în care un articol de date la care se face referire nu se află în cache se numește *cache miss*.
  - Dacă contribuția instrucțiunilor de transfer de date în timpul de execuție total al unui program este substanțială, un număr redus de ratări în cache va accelera semnificativ execuția programului.
  - O clasă evidentă de programe potrivite pentru o asemenea optimizare include programe care utilizează intens operațiile de bază pe tablouri.

Loop tiling

- Optimizare specifică efectuată prin optimizarea compilatoarelor C pentru a reduce numărul de ratări ale memoriei cache.
- Exemplu: Fie ciclurile:

```
for(i=0; i<m; i++) /* loop 1 */
        for(j=0; j<n; j++) /* loop 2 */
                if(i==0) b[j]=a[i][j]; else b[j]+=a[i][j];
```

care calculează suma rândurilor matricei a de dimensiune m x n

- Observatii:
  - Elementele de date accesate de referința b [j] sunt utilizate în mod repetat prin iterații succesive ale buclei 1.
  - Dacă n iterații ale buclei 2 sunt suficient de mari, elementele de date pot fi spălate din cache până în momentul utilizării lor repetate.
  - Pentru a minimiza înroșirea elementelor de date utilizate în mod repetat, numărul de iterații ale buclei 2 poate scădea.
  - Pentru a menține neschimbat numărul total de iterații ale acestei bucle, se introduce o buclă de control.
- Ca urmare, ciclurile transformate arată după cum urmează:

```
for(k=0; k<n; k+=T) /* additional controlling loop 0 */
        for(i=0; i<m; i++) /* loop 1 */
                for(j=k; j<min(k+T,n); j++) /* loop 2 */
                        if(i==0) b[j]=a[i][j];    else b[j]+a[i][j];
```

Această transformare se numește tiling, iar T este dimensiunea.

- îmbunătățește localitatea temporală a ciclurilor imbricate prin scăderea nr. iterațiilor între elemente de matrice utilizate în mod repetat.