Topic 14: Parallelism

COS 320

Compiling Techniques

Princeton University
Spring 2018

Prof. David August
Final Exam!

- Thursday May 3 in class
- Closed book, closed notes
Moore’s Law

Transistor Count

Source: Intel/Wikipedia
Single-Threaded Performance Not Improving
Decoupled Software Pipelining
Decoupled Software Pipelining (DSWP)

A: while (node)
B: ncost = doit (node);
C: cost += ncost;
D: node = node->next;

Inter-thread communication latency is a one-time cost
Implementing DSWP

DFG

A: r1 = M[r1]
B: r2 = r1 + 4
C: r3 = M[r2]
D: r4 = r3 + 1
E: M[r2] = r4
F: p1 = r1 != 0
G: br p1, Loop

L1:

SPAWN(Aux)
A: r1 = M[r1]
PRODUCE [1] = r1
F: p1 = r1 != 0
G: br p1, L1

Aux:

CONSUME r1 = [1]
B: r2 = r1 + 4
C: r3 = M[r2]
D: r4 = r3 + 1
E: M[r2] = r4
Optimization: Node Splitting
To Eliminate Cross Thread Control

L1
A: r1 = M[r1]
PRODUCE [1] = r1
F: p1 = r1 != 0
PRODUCE [2] = p1
G: br p1, L1

L2
CONSUME r1 = [1]
B: r2 = r1 + 4
C: r3 = M[r2]
D: r4 = r3 + 1
E: M[r2] = r4
CONSUME p1 = [2]
G’: br p1, L2
Optimization: Node Splitting To Reduce Communication

A: \( r1 = M[r1] \)
B: \( r2 = r1 + 4 \)
C: \( r3 = M[r2] \)
D: \( r4 = r3 + 1 \)
E: \( M[r2] = r4 \)
F: \( p1 = r1 \neq 0 \)
F": \( p1 = r1 \neq 0 \)
G: \( \text{br } p1, \text{ L1} \)
G": \( \text{br } p1, \text{ L2} \)

CONSUME \( r1 = [1] \)

register
control
memory

\( \rightarrow \) intra-iteration
\( \rightarrow \) loop-carried
Constraint: Strongly Connected Components

Consider:

- SPAWN(Aux)
  - A: \( r_1 = M[r_1] \)
  - B: \( r_2 = r_1 + 4 \)
  - C: \( r_3 = M[r_2] \)
  - PRODUCE [1] = \( r_3 \)
  - CONSUME [10] = [2]
  - F: \( p_1 = r_1 \neq 0 \)
  - G: \( br \ p_1, \ L_1 \)

Eliminates pipelined/decoupled property

Solution: \( DAG_{SCC} \)
Era of DIY:
- Multicore
- Reconfigurable
- GPUs
- Clusters

10-Core Intel Xeon
"Unparalleled Performance"
P6 SUPERSCALAR ARCHITECTURE (CIRCA 1994)

- Automatic Speculation
- Commit
- Parallel Resources
- Automatic Allocation/Scheduling
Multicore Architecture (Circa 2010)

- Automatic Pipelining
- Parallel Resources
- Automatic Allocation/Scheduling
- Automatic Speculation
- Commit
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ABCPL | CORRELATE | GUARD | HasL. | Haskell | HPC++ | Impact | Isis. | JAVAR. | Java RMI | JavaSpace | JIDL | Joyce | Khoros | Karma | KOAN/Fortran-S | LAM | Lilac | Linda | JADA | WWWinda | ISETL-Linda | ParLin | Eileen | P4-Linda | Glenda | Objective Linda | Occam | Omega | OpenMP | Orca | OOF90 | P++ | P3L | p4-Linda | Pablo | PADE | PADRE | Panda | Papers | AFAPI | Para++ | Paradigm | Parafrase2 | Paralation | Parallel-C++ | Parallaxis | ParC | ParLib++ | ParLin | Partams | Parti | pC | pC++ | PCN | PCP: | PH | PEACE | PCU | PET | PETSc | PENNY | Phosphorus | POET. | Polaris | POOMA | POOL-T | PRESTO | P-RIO | Prospero | Proteus | QPC++ | PVM | PSI | PSDM | Quake | Quark | Quick Threads | Sage++ | SCANDAL | SAM | pC++ | SCHEDULE | ScrlTl | POET | SDDA. | SHMEM | SIMPLE | Sina | SISAL. | distributed smalltalk | SML. | SONiC | Split-C. | SR | Streads | Strand. | SUIF. | Synergy | Telephos | SuperPascal | TCGMSG. | Threads.h++. | TreadMaks | TRAPPER | uC++ | UNITY | UC | V | ViC* | Visifold V-NUS | VPE | Win32 threads | WinPar | WWWinda | XENOOPS | XPC | Zounds | ZPL |
Parallel Library Calls

Realizable parallelism

Credit: Jack Dongarra
“Compiler Advances Double Computing Power Every 18 Years!”
– Proebsting’s Law
P6 SUPERSCALAR ARCHITECTURE

Spec-PS-DSWP

Core 1
LD:1
LD:2
LD:3
LD:4
LD:5

Core 2
W:1

Core 3
W:2
W:3

Core 4
C:1
C:2
C:3

P6 SUPERSCALAR ARCHITECTURE
Example
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
}

Program Dependence Graph

Control Dependence
Data Dependence
Spec-DOALL

Example

A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
}

Program Dependence Graph

Control Dependence
Data Dependence

Time

Core 1 | Core 2 | Core 3
---|---|---
A1 | B1 | C1
B2 | C2 | D2
Example
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
}

Program Dependence Graph

Control Dependence
Data Dependence

Core 1    Core 2    Core 3
A1    B1    C1    D1
A2    B2    C2    D2
A3    B3
Example

A: while (node) {
B: node = node->next;
C: res = work(node);
D: write(res);
}

Program Dependence Graph

- Control Dependence
- Data Dependence

Slowdown vs. Number of Threads

Spec-DOALL
**Spec-DOACROSS**
Throughput: 1 iter/cycle

Core 1 | Core 2 | Core 3
---|---|---
B1 → C1 → D1 → B4 → C4 → D4 → B7 → C7
C2 → D2 → B5 → C5 → D5 → C6
B3 → C3

**Spec-DSWP**
Throughput: 1 iter/cycle

Core 1 | Core 2 | Core 3
---|---|---
B1 → B2 → B3 → B4 → B5 → B6 → B7
C2 → C3
D2 → D3 → D4 → D5

Throughput: 1 iter/cycle
Comparison: Spec-DOACROSS and Spec-DSWP

Comm. Latency = 1: 1 iter/cycle
Comm. Latency = 1: 1 iter/cycle

Comm. Latency = 2: 0.5 iter/cycle
Comm. Latency = 2: 1 iter/cycle

Pipeline Fill time
Spec-DOACROSS vs. Spec-DSWP

[MICRO 2010]

Geomean of 11 benchmarks on the same cluster

Performance Speedup (X)

(Number of Total Cores, Number of Nodes)