Intro: Motivation [JIM DEMMEL]

Parallelism = only way to move forward in technology.

XSEDE = supercomputers NSF available to university

Paralab Project Berkeley

7 Dwarf.

Monte Carlos = Map Reduce

ASPIRE = new project: minimizing energy

117 faculty from 72 dept.

⇒ new minor for Ph.D.

Motivation for Parallelism

⇒ not enough data (e.g. weather ⇒ simulation)

⇒ too much data

the minor in computational science associated with pretty much all departments science + social science.

This course available for free from NSF + graded software + accounts at specific sites
Lawrence Livermore next to Berkeley
Hopper = 135,000 core machine

Applications: climate model = 2007 Nobel Prize
TeraGrid = old system; new project = XSede
Hello,

Please read this email in its entirety so as not to miss any important information:

The UC Berkeley 2013 Par Lab Boot Camp begins Monday, August 19 at 9am. You are registered for online attendance.

See the full schedule of lectures at:
http://parlab.eecs.berkeley.edu/2013bootcampagenda

To view a live web cast of the lectures during the event, please go to: mms://media.citris.berkeley.edu/parlab2013

Please note that the webcast link will be active ONLY during the event

Hands-on Assignments:

Online attendees are welcome to do the homework assignments using your XSEDE account. Information on registering for an account was sent to you separately. Please note that the deadline for registering for an XSEDE account was August 10.

Information and further instruction on the hands-on assignments will be given on Monday, August 19.

Detailed information on accessing and doing the hands-on assignments (as well as other logistics) can be found on the Logistics web page:
http://parlab.eecs.berkeley.edu/2013bootcamplogistics

Please contact: parlab-admin@eecs.berkeley.edu with any questions. Thank you for participating in the Par Lab Boot Camp 2013!

Par Lab Admin
Intro to Threads

John Kubiakowicz

100 processors in cars today

Parallelism

Many Core has arrived 2007: 80-core chip from Intel 100 million X河水

2010 cloud computer 24 tiles, 24-router mesh

Many Core? 64-core machine are here

How to program them

1) In video
   1) In user process
   2) In virus checking

2) Better use => parallel program

Modern programmers do not see assembly language

<table>
<thead>
<tr>
<th>Program</th>
<th>Int Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-File</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incl. set</td>
<td>X</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>Org. Tech</td>
<td>X</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>tech</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallelism is everywhere.

- 1 billion X factors all running in //
- within instruction ILP
- architects (until 2002) were trying to hide Vms from programmer, OS, & compiler
- pipeline @ example of wash -try & hold (or delay)
  @ Brand prediction target

⇒ need to worry about what goes on in different areas of parallel machine
  @ Data dependency
  ⇒ Need to fix data hazard
  @ Out -of -order instructions

⇒ SPECULATIVE EXECUTION

If wrong, need a way to unwind
⇒ interesting: use a lot of X factors and/or
⇒ speculative execution ⇒ waste
What has won?
-> smaller number of complicated processes.

Recent development: mobile phones have several processors, different complexity, different power.

More's law = # transistors.

35x’s law

This is why we need parallelism.
\[
\sqrt{LVW} \quad (??)
\]

**Vector code**

\[
\text{for } (i=0; i<64; i++)
\]
\[
C[i] = A[i] + B[i];
\]

**Vector version**

\[
\begin{align*}
L1 & : VLR,64 \\
LV & : V1,R1 \\
LV & : VR,R2 \\
ADDV & : V3,V1,V2 \\
SV & : V3,R3 \\
\end{align*}
\]

\[
\text{w} \times \text{of vectors}
\]

**GPUs (SIMD)** (nothing said on it!)

Nvidia chip w/ CUDA language

attached frame model
Concurrency at the level of thread
⇒ parallelism as tasks run in parallel

```plaintext
@cobegin
  job1(a1);
  job2(a2);
@coend

@tid1 = fork(job1, a1);
job2(a2);
join(tid1);
```

```plaintext
future
  v = future(job1(a1));  // could be done in //
  v = v;
```

Threads do not necessarily mean _PARALLEL_;
depend on hardware

PTHREADS = POSIX threading interface
⇒ STANDARD

Example of programming in threads.

Pitfall: too many threads!
⇒ 1 thread per core, n 2.

main
  A
  C
  B

Do not assume the threads will start in the order they were created.
Hardware multithreading

hyperthread = simultaneous multithreading

\[ \text{Thread 1} \]
\[ \text{Thread 2} \]
\[ \Rightarrow \text{simultaneous threads inside process} \]

Parallelism of 2 is better than 4 threads in 1 in CPU

Sequential memory system

\[ \text{Latency} \quad 1000 \]
\[ \text{CPU} \quad \text{gap} > 50\% \quad \text{pu year} \]
\[ \text{DRAM} \]
\[ 1980 \]
\[ 2000 \]

Power versus operator:

DP flop = 3
Communication = memory times \( \times 3 \)
Communication is expensive in terms of power.

Locking in shared memory.
- Example with sum. Sum up \( \frac{1}{2} \) and \( \frac{1}{3} \) numbers with 2 threads. Lock the sum variable.

Cache gets in the way because sum could be in different caches.
- Cache coherence will solve this.

Synchronization:
- Mutex (primitive in Pthreads)
- Barrier (primitive in Pthreads)
- Transactional Memory (?)

Message Passing

Explicit Communication:

MPI: old
- lots of overhead
- goal = portable

Shared Mem
- Implicit (won't need)

MPI
- explicit communication, overhead high
Sandy Bridge

CPU

4 Cores

Cache

Ring network/IOs
OPEN MP [Tim Mattson, Intel]

Parallel programming is hard. But programming is hard.

"Never write a parallel FFT!"

lots of framework around for creating parallel applications
- CUDA
- C++ Amp

- All programs written following just 7 design patterns
  - SPM
  - Kernel parallelism
  - Fork / Join
  - Actors
  - Vector
  - Loop parallelism
  - Work pile

Open MP: "let's make it as easy as we can."

⇒ Simple API

Best example for threads: computing $T$

\[
T = \int_0^1 \frac{1}{(1 + x^2)} \, dx
\]
take example + do it in Pthreads
use Fork-Join Pattern
all very repetitive => automate
this is OpenMP

Fork-Join pattern:

In OpenMP, main thread has ID 0
Example for f = SIMD

OpenMP = good way to learn parallelism.
Performance Tuning

[Garry Carlton, Intel] [boring guy]

Software profiling tool

Tool = VTune Amplifier X 2013
* Tuesday *
Message Passing MPI [Tim Matson]

- 200-300 function
- Most important paradigm of programming

* Top 500

\[ \text{Top} \quad \text{KernTop} \quad \text{20XX ?} \]

\[ \text{Cray} \]
Manively Parallel Processor, MPP
off the shelf processors + a lot of customized hardware

1996 ASCII Red first TeraTop computer
Clusters of off-the-shelf PCs late 90's and software Mercury Panning good

- Cosmic Cube
# systems

Single Proc

HPC Convergence

Cluster

HPC 1992 → Now
Sure of Parallelism and Locality in Simulation [Jim Demmel]

- Moving data (communication) is expensive
- Game of life

[Grid diagram]

Better solution
fewer cuts along perimeter

⇒ Graph partitioning problem
NP, but a lot of good tools exists

Asynchronous Simulation
Asynchronous more efficient

$\Rightarrow$

Conservative

Speculative: keeping best sometimes back up to known point

PARTICLE SYSTEMS

- Stars
- Particles
- etc (cars)

LIMITED VARIABLES

- Electrical Systems
ARCHITECTING PARALLEL SOFTWARE WITH PATTERNS

[kurt kentzer ucb]

Typical

initial code

profile

performance profile

fast enough

release

add threads

not fast enough

What is life w/o modularity?
- spaghetti code

* What computation we do is as important as how we do them.

Pattern: describes a problem that repeats over and over- originally about civil architecture
Patterns give a name to things
- give a palette of choices
- allows way to check if one has explored all options

1. Pip and Filter
   ![Diagram](https://example.com)

2. Iterator
   ![Diagram](https://example.com)
   - convergence
   - optimization

3. Map Reduce
   * functional programming

4. Linear Algebra (vector, matrix)
* Spectral methods (FFT)
* Dynamic Programming

13 patterns total
Cuda [Bryan Catanzaro]

Sandy Bridge: 8 cores, 2 issues, 3.6 GHz
- 2 threads/core
- 96 strands, 16 GiB bandwidth
- Kepler: 14 SMS, 750 MHz
- 6 issues
- 28,672 threads at same time
- 250 GB/s

CPU

\[ \text{CPU} \]

\[ \text{GPU} \]

\[ \text{CUDA} \]

SIMD
- Neglected
- Intel's new SIMD architecture was MMX and now is AVX
- Not using SIMD in a current Intel process is almost throwing away 96% of performance possibility

\[ \Rightarrow \text{CUDA} \]
Cuda in C

Thread blocks: blocks that can execute in any order, sequentially or in parallel.
Unified Parallel C [Wally Gillett]
(no streaming)

Computational Pattern [Jim Demmel]

large power per
operation is measured:
- Joules/inst, Joules/message
- Joules/storage, Joules/leakage

some algorithms which behave
ideally will run 8 as fast requiring
2 as much energy.

Performance Debugging [David Skiena]

NERSC @ LBL
- Pef & Scalability
- Tools for debugging.

CrayPat ➔ Support C, C++, UPC, Threads, OpenMP, MPI, etc...
- Tools use Sampling & Tracing
MAP REDUCE [Matti Zaharia, UCB]

Facebook collects ~50 TB data daily
Cost TB ~ $50 per customer
$20 per company

50 MB/s bandwidth => 1 TB read = 6 hrs
Failure -> as # of proc ->

=> MR message passing becomes hard as scale ->

=> Data parallel runs attractive

=> MapReduce (2004 paper by Google)

=> Yahoo created Hadoop and by all Companies except Google or Microsoft

Data type [key: value]
Map (k, v) => list (k, v)
Reduce (k, list (v)) => list (k, v)

def mapper(line):
    for each word: line. split()
    output (word, 1)
Hadoop

- unique maps/tasks to nodes
- does fault recovery
  - if randomness is used by a task, the random seed should be the same or reproducible so that task can be restarted easily

Example

- search output lines matching pattern
- sort
- inverted index
- find top 100 most frequent words
multiple stages will be used to solve complex problems.

- Java API
- Python (Hadoop Streaming)
- Pipes (C++)

Demo: SPARC

Since 2009
use Python maps & reduce in a new API.
Fast short coding (5 lines for word count)