Towards Parallel Machines for Everyday Use

Fall Research Forum
October 14th, 1993

Thorsten von Eicken
Department of Computer Science
Cornell University
Why Parallelism? (Technology Trends)

Workstation performance

- Floating point Spec: 75% per year
- Integer Spec: 50% per year

Clock cycle

- “Doubling every two years”
- (i.e. 40% per year)

Where do the remaining 10%–40% come from?
Technology Trend: Transistor Counts

Transistor counts: “4x every 3 years”
- i.e. 60% per year

© T. von Eicken, 1993
Performance Trend Explanation

Performance equation:
- \( \text{time} = \text{instructions executed} \times \text{cycles per instruction} \times \text{clock cycle} \)

Performance improvements:
- clock cycle: hw technology
- instructions executed: compilers
- cycles per instruction: parallelism

Parallelism used today:
- bits per cycle
  - bus/registers/alu width
  - multiply, divide bits per cycle
- memory accesses per cycle
  - multi-port register file
  - split I&D cache
- instructions per cycle
  - pipelining
  - superscalar

=> how much such “invisible” parallelism is left for the next round of tuning? And thereafter?
The Microprocessor of 2,000

- 16-20 M Transistors
- MByte SRAM
- 8-16 Instruction Units
- 4-8 FPUs
- Extremely high BW mem interface
- Multi GB/s network interface

-> Today's medium-scale multiprocessor system!
Observations about Multiprocessors Today

Every high-performance system manufacturer:
- builds small-scale (4-8 nodes) bus-based multiprocessors
- announces medium-scale (16-32 nodes) “bus”-based multiprocessors
- is putting a large number of “workstations” on a network

None of these processors have a network interface!
- No consensus on this fundamental aspect of Parallel Architecture
- No definitive software base
- No market

Multiprocessor nodes cost 3x equivalent workstation
- network and support is expensive
- no market

How does this lead to the µmulti of 2000?
Questions

Why is there no market?
- no software
- MP node costs 3x equivalent workstation

Why is there no software?
- the machines are hard to program&use
- no market

Why are they so hard to program?
- inadequate parallel architecture
  - leads to poor language, compiler, OS

=> The Goal of Parallel Architecture is to extend commodity microprocessor systems to be cost-effective building blocks for multiprocessors and to enable a healthy software market.
by the way... Parallel Processing is Widely Applicable

Grand Challenge Problems

Global Change
Human Genome
Fluid Turbulence
Vehicle Dynamics
Ocean Circulation
Viscous Fluid Dyn’s
Superconductor Mod.
QCD, Vision

Mem
1 TB
100 GB
10 GB
1 GB
100 MB
10 MB

48 Hour Weather
3D Plasma Modelling
Oil Reservoir Modeling
2D Airfoil
Vehicle Signature
72 hour Weather

Structural Biology
Pharmaceutical Design
Chemical Dynamics


100 Mflops 1 Gflops 10 Gflops 100 Gflops 1 Tflops

... what about grand $$$ problems?
Parallel Architectures are Converging

Nodes are complete computers
- off-the-shelf processors
- workstation technology
- NEW: network interface

Networks
- low-latency
- high-bandwidth
- carry messages
- packet-switched

message passing
message driven
shared memory

dataflow
SIMD
The State of the Art

Force the network into an existing interface and develop a programming model that reflects the resulting structure.

- Memory Interface => Shared-Memory
- I/O Interface => Message Passing / CSP
- ALU => Array Operations

Are these traditional structures / styles...
- natural?
- compelling?
- optimal?

A Metric of Evaluation: does the architecture...
- facilitate or frustrate compilation? (support powerful programming models?)
- allow the underlying resources to be fully utilized?
Outline

Why parallel computing?

Critique of Message Passing

Critique of Shared Memory

Potential Alternatives
General Beliefs about Message Passing

Style

P1 send a, P2

P2 receive P1, b

Structure

Network

P-M P-M ⋯ P-M

Hard to program
• no global address space

Hard to understand programs
• much of the algorithm is implicit in the communication pattern.

Efficient communication primitive
• “it’s so simple: it must be efficient”

Easy to support in hardware
• ditto...
Understanding Networks

Mind-set:

\[ P1 \rightarrow P2 \]
send a, P2

\[ P2 \rightarrow P1, b \]
receive P1, b

Real network:

Send Overhead

\[ O \]
injecting

W

Routing

H \cdot R

Receive Overhead

\[ O \]

Performance

- O - 100µs - 1000’s of cycles
- H \cdot R - 10µs - 100’s of cycles
- I - 0.1µs/byte - 1’s of cycles
Resource Utilization (Non-Blocking Send&Receive)
### Send/Receive Overhead

<table>
<thead>
<tr>
<th>Machine</th>
<th>Ts+Tr</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPSC</td>
<td>4,000µs</td>
<td>???</td>
</tr>
<tr>
<td>nCUBE10</td>
<td>400µs</td>
<td>4,000</td>
</tr>
<tr>
<td>iPSC/2</td>
<td>700µs</td>
<td>11,200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(350µs below 100 bytes)</td>
</tr>
<tr>
<td>IBM SP-1</td>
<td>200µs</td>
<td>12,500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(30µs without protection)</td>
</tr>
<tr>
<td>nCUBE/2</td>
<td>150µs</td>
<td>3,000</td>
</tr>
<tr>
<td>iPSC/860</td>
<td>160µs</td>
<td>6,400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(60µs below 100 bytes)</td>
</tr>
<tr>
<td>CM-5</td>
<td>95µs</td>
<td>3,200</td>
</tr>
<tr>
<td>Delta</td>
<td>60µs</td>
<td>2,400</td>
</tr>
</tbody>
</table>
Message Passing as Compilation Target

Fortran-D

FORALL . . .

Send, Send, . . .
Compute local

Rec, Rec, . . .
Compute

Barrier

Assumes unbounded buffering
Requires decomposition function and its inverse.
But: compiler knew destination address!
And storage was pre-allocated!
Outline II

Why parallel computing?

Critique of Message Passing

Critique of Shared Memory

Potential Alternatives
General Beliefs about Shared Memory

Style

Easier to program
• global address space
• migration path from conventional programs

Hard to understand programs
• because interactions between processors are unstructured (e.g., races).

More complex hardware support
Cost-Effectiveness

+ Lower overhead (1 - 2 µs, 30-50 cycles)
  – Fixed-size transfer unit
  – Processor stalls during communication, network idle during computation
+ Potential for automatic management of data placement
  – Doesn’t recognize there’s a processor at remote memory
    • co-ordination via many memory operations: (lock, update, unlock)
Outline III

Why parallel computing?

Critique of Message Passing

Critique of Shared Memory

Potential Alternatives
Active Messages

Idea:
Associate a custom handler with each message

Head of the message is the address of its handler

Handler executes immediately upon arrival
- extracts msg from network and integrates it with computation, possibly replies
- handler does not ``compute´´

Advantages:
- versatility: can implement variety of communication protocols using handlers
- efficiency: can use pre-allocated storage, pass addresses in messages,...
Power of Active Messages

<table>
<thead>
<tr>
<th>Machine</th>
<th>Send Time (µs)</th>
<th>Receive Time (µs)</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM-5</td>
<td>1.6</td>
<td>1.7</td>
<td>(21 instructions)</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>13</td>
<td>(26 instructions)</td>
</tr>
</tbody>
</table>

Current programming models built on (CM-5) Active Messages:

- **Split-C**
  - Split-phase access to distributed memory
  - Achieves full machine utilization with reasonable effort
- **Implicitly parallel language ld90** (MIT Dataflow community)
  - Good performance on large unstructured programs
  - Compiler controlled multithreading to hide latency
- **Send&receive message passing**
  - 5x vendors CMMD for short messages (20µs overhead)
- **Shared memory**
  - Research test-bed

Vendor’s MIMD communication library built on Active Messages
Split-C: Engineering Parallel Programs

Mind set:

LogP model
- Simple Compiler
- Address Arithmetic
- Instruction Scheduling
- Routine optimizations

Active Messages
Simple Hardware

Global Address Space
LogP

Shared Memory
CSP
Data Parallel

L: message latency
o: message overhead
g: message gap
P: processors

Remember RISC:
- Keep hardware simple and fast
- Simple, predictable performance model
- Optimize software
Split-C: Explicit Access to Global Memory

Local Memory

Non-Local Memory

Cache

Regs

...
Split-C in a nutshell

Optimize remote access in a global address space

2-D Global Address Space
- physically distributed memory
- no specified topology
- double *global P;
  x = *P

Split-phase access
- overlap communication
  x := *P

Bulk transfer
- struct {...} *Q;
  foo x = *Q;

Distributed Arrays
- double array
  A[n][m][k][k];

Signalling Stores
- P := y;
Conclusions

The parallel processing "Revolution" is finally happening
• The underlying technology supports it and demands it.
• It is the only way of maintain a credible product line.

For it to happen, a few billion dollar software investment is required

This will only occur if we get the architecture right
• Need a large base of inexpensive high-performance MPPs

A meaningful language base is essential

Time is extremely short.
• Technology is racing on
• Software developers need to revise or rewrite once!