STAMP: A Universal Algorithmic Model for Next-Generation Multithreaded Machines and Systems

Michel Dubois, Hyunyoung Lee, Lan Lin
Motivations (1)

• The trend in micro-architecture today is towards multiple cores in shared-memory configurations
• Each core is itself multithreaded
• So that future microprocessors will execute many threads in parallel
• In the future, Moore’s law applied to computing will be maintained by doubling the number of threads ever other year
  • Not by increasing the frequency
Motivations (2)

- Increasing frequency is untenable because of power issues.
- CMPs will then be connected in shared-memory or message passing configurations.
- High-end applications running on high-end servers such as HPC and database systems have abundant number of threads and will thrive in such environment.
- We need tools to quickly develop algorithms that can scale on future desktops (workstations and PCs) with ever increasing number of threads.
Motivations (3)

• The tool must
  • Estimate performance and scalability
  • Estimate power
  • Model shared memory and message passing
  • Be simple enough to be quickly deployed
  • Be reasonably accurate at least enough to estimate scalability
  • Capture essential structure and parameters of parallel applications
  • Include new emerging programming paradigms, such as transactional memory
Multithreaded Machines

Example: Niagara multicore chip
Examples of Traditional Parallel Computational Models

- PRAM
- QSM (Queued Shared Memory)
- BSP (Bulk Synchronous Parallel)

- All existing models are too restrictive!
- Power is not incorporated!
STAMP

- Synchronous, Transactional, and Asynchronous Multi-Processing
- Generic algorithmic model (parallel, distributed & nested processes that cooperate w/ each other)
- Universal performance & power complexity model
STAMP Process

- A STAMP process
- An execution: A sequence of local computations & communication operations
- A local computation: Operation that can be performed in a single processing unit
- A communication operation
Distribution of STAMP Processes

- Intra-processor
- Inter-processor
- Trade-offs between execution time (performance) complexity and power/energy complexity
## Synchrony in STAMP

Possible combinations of modes of execution & communication based on synchrony

<table>
<thead>
<tr>
<th>Exec</th>
<th>Transactional</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong></td>
<td>[trans_exec] [synch_comm]</td>
<td>[async_exec] [synch_comm]</td>
</tr>
<tr>
<td><strong>Asynchronous</strong></td>
<td>[trans_exec] [async_comm]</td>
<td>[async_exec] [async_comm]</td>
</tr>
</tbody>
</table>
Distribution & Synchrony in STAMP

Jacobi(A[], b[], x) [intra_proc, async_exec, synch_comm]
bool terminated=false; int t=0
while not terminated
  forall i: x_i(t+1)=-1/a_{ii}[\sum_{j\neq i} a_{ij}x_j(t)-b_i]
  if (termination cond is met) terminated=true

Example using attributes of distribution, execution and communication – solving a system of linear equations (Ax=b) using the Jacobi algorithm
Structure of STAMP

Distributed Jacobi algorithm for process i

```
bool terminated=false; int t=0
while not terminated
    receive x(t) from all other processes
    xi(t+1)=-1/aii[Σj≠i aijxj(t)-bi]
    send xi(t+1) to all other processes
    /* implicit barrier synchronization here*/
    if (termination cond is met)
        terminated=true
```
A STAMP Algorithm

- Consists of any combinations of
  - S-units
  - Nested STAMPs
  - Parallel/Distributed STAMP processes
STAMP Complexity Models

- **Performance (execution time) complexity**
  - In one time unit a local operation can be computed by a processing component on data available in memory local to it
  - For S-round and S-unit: add the time needed for local operations, shared memory accesses and message exchanges
  - For parallel/distributed STAMP processes: take the maximum among the execution times of them

- **Power/energy complexity**
  - Add the energy of each computation, shared memory access and message exchange
Parameters (1)

- **Global parameters:** $P_a, P_e, n$
- **For local execution:** $c_{fp}, c_{int}, w_{fp}, w_{int}, c$  
  (c: time cost for local computations including fp & int operations)
- **For communication**
  - For shared memory access
  - For message passing
Parameters (2)

- For shared memory
  - $l$: upper bound on delay in accessing a shared memory module due to memory hierarchy
  - $k$: maximum number of accesses to any shared memory location
  - $g_{sh}$: bandwidth defined as the ratio of the # of local operations performed by the thread in one time unit to the total number of memory accesses in the same time unit
  - $d_r, d_w$: #s of shared memory read and write ops
  - $w_{dr}, w_{dw}$: energy per shared memory read and write
Parameters (3)

- For message passing
  - $L$: upper bound on message delay
  - $g_{mp}$: bandwidth defined as the ratio of the number of local operations performed by the thread in one time unit to the total number of messages delivered in the same time unit
  - $m_s, m_r$: numbers of message send and receive operations
  - $w_{ms}, w_{mr}$: energy per message send and receive
Complexity Measures (1)

- For an S-round
  - $T_{S\text{-round}} = c + [\text{shared memory comm}] (k + [P_e \geq 1]l_e + [P_a \geq 1]l_a + g_{sh\_a}(d_{r\_a} + d_{w\_a}) + g_{sh\_e}(d_{r\_e} + d_{w\_e}))[\text{message passing comm}] ([P_e \geq 1]l_e + [P_a \geq 1]l_a + g_{mp\_a}(m_{s\_a} + m_{r\_a}) + g_{mp\_e}(m_{s\_e} + m_{r\_e}))$

- $E_{S\text{-round}} = c_{fp}\_w_{fp} + c_{int}\_w_{int}$
  + $w_{dr}(d_{r\_a} + d_{r\_e}) + w_{dw}(d_{w\_a} + d_{w\_e})$
  + $w_{mr}(m_{r\_a} + m_{r\_e}) + w_{ms}(m_{s\_a} + m_{s\_e})$
Complexity Measures (2)

- For an S-unit
  - $T_{S\text{-unit}} = \sum_{\text{all } S\text{-rounds}} T_{S\text{-round}} + T_c$
  - $E_{S\text{-unit}} = \sum_{\text{all } S\text{-rounds}} E_{S\text{-round}} + E_c$
  - $P_{S\text{-unit}} = E_{S\text{-unit}} / T_{S\text{-unit}}$

- For a STAMP proc. with more than one S-unit: sum of all $T_{S\text{-unit}}$ or $E_{S\text{-unit}}$

- For parallel/distributed STAMP proc.s: max execution time, total energy/power
Example (revisited)

Distributed Jacobi algorithm for process $i$

```plaintext
bool terminated=false; int t=0
while not terminated
    receive $x(t)$ from all other processes
    $x_i(t+1) = -1/a_{ii} \left[ \sum_{j \neq i} a_{ij} x_j(t) - b_i \right]$
    send $x_i(t+1)$ to all other processes
    /* implicit barrier synchronization here*/
if (termination cond is met) terminated=true
```
Bounds on Complexity Measures

- \( T_{S\text{-unit}} \geq 2n + 5 + 2n \{\frac{3}{(n(n-1))}\} - 2\{\frac{3}{(n(n-1))}\} + 2 \)
  \( = \frac{2n + 6}{n + 7} \geq 2n \)
- \( E_{S\text{-unit}} \leq (2(x+y)w_{\text{int}})n \)
- \( P_{S\text{-unit}} \leq \frac{(2(x+y)w_{\text{int}})n}{2n} = (x+y)w_{\text{int}} \)

- Assume: every core has the same power limit of \( 3(x+y)w_{\text{int}} \), then the Jacobi algorithm should not be assigned to more than three intra-processor threads per core.
Conclusions

• Future microarchitectures will be multithreaded
• We need effective tools to design scalable algorithms quickly
• Here, we proposed a new generic algorithmic model called STAMP
• Encompasses synchronous, transactional, and asynchronous computation and communication models
• Equipped with universal performance and power complexity models
• Illustrated how to design and analyze algorithms using STAMP, and how to apply the complexity estimates to better utilize CMP/CMT-based machine within given constraints such as power
Future Works

- Simulations
- Implementations
- Evaluation & validation
Thank you!

Any Questions?