Parallel programming with Session Java

Nicholas Ng (nickng@doc.ic.ac.uk)

Imperial College London
Motivation

- Parallel designs are difficult, error prone (eg. MPI)
- Session types *ensure* communication safety in concurrent systems
- So use session types to design *safe* parallel algorithms for high performance clusters
Contributions

- An implementation of parallel n-body simulation
  1. Programmed in **Session Java (SJ)**, a full implementation of session types
  2. Uses **FPGA** on the AXEL heterogeneous cluster

- A formal description of **multicast outwhile, inwhile** SJ primitives in session types

- Showed type soundness, progress property in SJ parallel programs connected in a ring topology

- Proved SJ n-body implementation deadlock free
Session types

- Typing system for $\pi$-calculus
- $\pi$-calculus models structured interactions between processes
- Main idea: communication primitives should have a dual

Example

(Conventional type system) \texttt{int } i = 9

- $i$ is type \texttt{int}
- $9$ is type \texttt{int}

Process A: $c_{ab}!\langle 9 \rangle; P$ (send 9 to B via channel $c_{ab}$)
Process B: $c_{ab}?(x).Q$ (receive $x$ from A via channel $c_{ab}$)

- A is type Send \texttt{int} (or $c_{ab}: ![\texttt{int}]$)
- B is type Receive \texttt{int} (or $c_{ab}: ?[\texttt{int}]$)
Session programming with SJ

Session Java (SJ) [HYH08]

- A full implementation of binary session types in Java
- Provides a socket programming interface with eg. `accept()`, `request()`, `send()`, `receive()`

Workflow of a SJ program:

1. Declare session type/protocol of program in SJ
2. SJ compiler checks local session type conformance
3. Runtime duality check with communicating program
SJ features for parallel programming

- Iteration chaining
- Multi-channel `inwhile` and `outwhile` in place of reduce-scatter operations

**Master:** `<s1,s2>.outwhile(i<42){...}`
**Forwarder1:** `s3.outwhile(s1.inwhile){...}`
**Forwarder2:** `s4.outwhile(s2.inwhile){...}`
**End:** `<s3,s4>.inwhile(){...}`
Simple example: N-body simulation

- $n$ particles following Newton’s laws of motion
- Calculate the result force acting on each particle
- Displace the particle based on net force acting on it

**Figure:** Result force is vector sum of all forces
Simple example: N-body simulation

- Implemented in a *ring* topology
- 3 kinds of processes - Master, Worker (multiple), LastWorker

1. Each allocated a partition of particles
2. Calculate resultant forces on received set of particles
3. Forward to next node
4. Repeat until end of one time step
Another example: Jacobi method

- Iteration-based method for solving the Discrete Poisson Equation
- Used in physics and natural sciences
- Given initial prediction, iterate until converged or upper limit of iterations

<table>
<thead>
<tr>
<th></th>
<th>edge</th>
<th>value</th>
<th>edge</th>
<th>value</th>
<th>edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge</td>
<td>value</td>
<td>value</td>
<td>edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge</td>
<td>value</td>
<td>value</td>
<td>edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>edge</td>
<td>edge</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure: A sub-matrix of calculation
Another example: Jacobi method

- Implemented in a *mesh* topology (2D decomposition)
- 9 kinds of processes - one for each edge case and a Worker in the center

1. Each allocated a sub-matrix of values
2. Calculate average of neighbouring values for all element
3. Exchange edges to adjacent sub-grid
4. Repeat until converged
AXEL: a heterogeneous cluster

Axel [TL10] is a heterogeneous cluster that contains different *Processing Elements* (PE) on each node:

**CPU**  Off-the-shelf *multicore* x86 architecture CPU

**GPU**  *Graphics Processing Unit*, nVIDIA Tesla, dedicated General Purpose GPU

**FPGA**  *Field Programmable Gate Arrays*, reconfigurable hardware

- AXEL is a 16-node NNUS cluster
- Each node can be used as individual PC
- Connected by high speed Ethernet
Performance benchmark results

- Against MPJ Express [SCB09], implementation of MPI in Java
- Performance competitive (Left: N-body simulation, Right: Jacobi method)
Performance benchmark results (with FPGA)

- Better performance with more particles
- Best performance: SJ+FPGA 2x faster than SJ implementation
Well-formed topology

- Multichannel `inwhile` and `outwhile` not safe on its own
- Well-formed topology: Topology constructed as DAG with 1 root node and 1 `sink` node
- Individual pairs of sessions are dual
- Iteration controlled by a single condition in the Master node
- Deadlock freedom for group of processes in well-formed topology
Future (and ongoing) work

C based language implementing session types

- Higher performance with FPGA or other acceleration hardware
- Can integrate with AXEL or similar HPC applications toolchain
References

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo.
Language primitives and type disciplines for structured communication-based programming.

Raymond Hu, Nobuko Yoshida, and Kohei Honda.
Session-based distributed programming in java.

Aamir Shafi, Bryan Carpenter, and Mark Baker.
Nested Parallelism for Multi-core HPC Systems using Java.

Kuen Hung Tsoi and Wayne Luk.
Axel: a heterogeneous cluster with FPGAs and GPUs.