Kathleen Feng, Ang Li  
27 September 2017

PRAM

Theoretical model to analyze parallel algorithms, key resource is number of memory access
Turing Machine - simple enough to theorize about. Cannot capture parallelism.

Differences between Fortune/Wyllie and CLRS
- Both don’t take into account computation costs, focus on concurrent memory access/cycles
  - PRAM does poor job of encompassing running time if computation work is order of magnitude larger than memory accesses (e.g. $O(n^3)$ vs $O(n)$)
- Fortune does not have bound on processors
- Fortune does not have current write (CREW)

Problems with PRAM models
- Both do not mention cost for communication
  - Physics laws say information cannot be conveyed within zero time across some non-zero distance.
- Synchronization cost
  - CLRS: inexpensive barrier synchronization mechanism, not completely flawed when $n$ is big
  - Example: barrier on $n$ processors with only pairwise communications => $O(\log n)$

CLRS models:

<table>
<thead>
<tr>
<th>Read/Write (to the same address)</th>
<th>Exclusive</th>
<th>Concurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive</td>
<td>EREW</td>
<td>ERCW</td>
</tr>
<tr>
<td>Concurrent</td>
<td>CREW</td>
<td>CRCW</td>
</tr>
</tbody>
</table>

Ranking from easy to hard (hardware):
1. EREW
2. CREW, ERCW
3. CRCW (tied to cache coherence, consistency models)
   a. Priority write usually found

Note: EREW means programmer must make sure concurrent read/write doesn’t happen, and if it happens, the machine will halt and catch fire (behavior not defined).

PRAMs good for asymptotic analysis, most popular one, theorists use this framework

StreamIt
Is it a language? Embedded a language inside a language
20 or 30 papers on StreamIt
Did they change their syntax? No, stuck with this
DSL vs embedding in current language
StreamIt programs can be executed sequentially. The order is known.

Is this parallel? Expressing some type of parallelism
Pipeline parallelism
SplitJoin: data-level parallelism
Explicit or implicit? Very implicit, program is not telling where the parallelism is
  • SplitJoin can be run sequentially
Optimization: filter fusion, etc.

Streaming applications characteristics:
  1. Large streams of data
  2. Independent stream filters
     o Are they necessary? Maybe not, according to Cray
  3. Stable computation pattern
     o Perhaps ill-defined
  4. Occasional modification of stream structure: not everybody would agree with that
  5. Occasional out-of-stream communication: some people think only affine transforms
  6. High performance expectations: mentions about real time

Could be used for server pipelining (AWS)
Don’t declaratively describe graph, programmatically inflate graph with worker functions
  • Can have loops
  • Some argue that it is bad, Ptolemy language split declaratively describing graph from worker functions
  • Compiler might not be able to make optimizations
  • Used symbolic execution in compiler to figure out graph
  • Order matters when adding filters
  • Timing comes in with feedback and changing graph due to outside actors

Performance compared against SpectrumWare/hard-coded C implementation
SpectrumWare still used today in cell phone towers, no compiler, just a library approach (Vanu Inc.)

Not as good as C code, beats SpectrumWare in some cases
Got StreamIt to be much faster with compilers later on
Fair amount of impact from language perspective

Main contribution of paper:
  • Formalization of problem
Language

Accept this paper into a conference? Wouldn’t get into programming languages venue