Memory Feasibility Analysis of Parallel Tasks Running on Scratchpad-Based Architectures

Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo

+ Scuola Superiore Sant’Anna, Pisa, Italy
* CISTER, ISEP, Polytechnic Institute of Porto, Portugal
• **Focus of this talk:** multicore platforms with local memories (scratch-pads)
Are the local memories **large enough to satisfy** the maximum **memory space requirement** generated by the application?
This Talk

Application

Partitioning Algorithm

Timing Analysis

Memory Feasibility (This paper)

Core 1
LM 1
Global Memory
Crossbar

Core 2
LM 2

Core 3
LM 3

Core 4
LM 4

Application

Feasibility (This paper)

Application Partitioning Algorithm Timing Analysis
Objective of this Paper

This paper: How to bound the memory space requirement of parallel tasks?
Task Model

- Directed Acyclic Graph (DAG)
- Each node is **statically assigned to a processor** and executed in a **non-preemptive** manner
Task Model

- Directed Acyclic Graph (DAG)
- Each node is statically assigned to a processor and executed in a non-preemptive manner

Edges denote precedence constraints and communications. **Weights** denote the amount of data exchanged by nodes *(bytes, or blocks, or memory pages, ...)*
Task Model

- Directed Acyclic Graph (DAG)
- Each node is statically assigned to a processor and executed in a non-preemptive manner

Inter-core communication realized with shared buffer in global memory

CPU 0

v1  v2  v5

CPU 1

v3  v4  v6
Task Model

- Directed Acyclic Graph (DAG)
- Each node is \textit{statically assigned to a processor} and executed in a \textit{non-preemptive} manner

\textbf{Intra-core} communication may be realized with a shared buffer in \textit{local memory}.
Memory Space Requirement: Nodes

- During their whole execution, nodes must dispose of all input and output buffers allocated in the scratchpad of their core.
- Nodes are also characterized by a local memory space requirement (e.g., to model the stack of functions).
Memory Space Requirement: Nodes

- **Intra-core communications**
  - Input buffers are left allocated by predecessors
  - Output buffers are allocated when the node starts executing

- **Inter-core communications**
  - Input buffers: copy-in phase from global memory to scratchpads
  - Output buffers: copy-out phase from scratchpads to global memory
What’s the effect of precedence constraints on memory space requirement?
Memory Space Requirement

Graph with nodes v1, v2, v3, v4, v5, v6 and edges with labels 4, 2, 3, 2, 3, 0, 0.

Bar chart for CPU 0 and CPU 1 with memory space requirement on the y-axis and time on the x-axis.
Memory Space Requirement

CPU 0

CPU 1

Memory space req. CPU 1

3+2+0+1=6

-4

time

communication completed
Memory Space Requirement

Diagram with nodes v1, v2, v3, v4, v5, v6 and edges 3, 2, 4, 2, 3, -5, and 8. The graph shows dependencies and memory space requirements.

Memory space req. CPU 1:
- 2 + 2 + 3 + 1 = 8
- -5

CPU timeline:
- CPU 0: steady
- CPU 1: 2 intervals
Memory Space Requirement

CPU 0

CPU 1

Memory space req. CPU 1

3+1=4

time
Objective of this Paper: a closer look

This paper: how to compute an upper-bound on the maximum memory space requirement generated on each core

Memory space req. CPU 1
Objective of this Paper: a closer look

Bounding the **maximum memory space requirement** in the presence of **complex** precedence constraints is **not trivial**...

Automotive software (complex inter-runnable dependencies)

Deep neural networks
PROPOSED SOLUTIONS
• This work presents **two approaches** to compute a bound on the memory space requirement generated by a set of parallel tasks

\[ f(x) = ... \]

**Closed-form** bound via a top-down decomposition of the memory space requirement

**Algorithmic** methods based on

• **max-plus algebra** for **fork-join tasks** (very efficient)
• mapping to a **max-flow cut** problem for arbitrary **DAG tasks** (solvable with both MILP or graph transformations)
Top-down decomposition of the memory space requirement

- For each task...
- Each task may be executing or not...
- Due to non-preemptive scheduling, when it is executing there is only one node started but not completed...

\[
\begin{align*}
M^\text{EX}_i &= \max_v \{M^\text{NODE}_{i,v} + M^\text{INTRA}_{i,v}\} \\
M^\text{NEX}_i &= \ldots
\end{align*}
\]

Approximations (difficult to bound)
The memory space requirement of fork-join tasks can be computed with a max-plus algebra (for nodes that are not executing).

- The three communications can be pending at the same time.
- The two communications cannot be pending at the same time (serialized).

Algorithmic Methods: Max-plus Algebra
• **Efficient solution** for fork-join tasks

• By repeatedly applying the two rules, it is possible to express the memory requirement with a **formula**, which, for instance, may be useful to implement a **sensitivity analysis**
Algorithmic Methods: Max-flow Cut

- Applies to arbitrary **DAG** structures
- Consider a node $v$. At any time, due to non-preemptive execution, all the other nodes can either be
  - completed; or
  - not yet started

![Diagram of a DAG with nodes labeled and edges marked with weights. The diagram shows a cut with completed nodes.]
Algorithmic Methods: Max-flow Cut

- Applies to arbitrary DAG structures
- Consider a node $v$. At any time, due to non-preemptive execution, all the other nodes can be:
  - completed; or
  - not yet started

Edges on the cut represent communications pending at the same time.
Algorithmic Methods: Max-flow Cut

- Finding the maximum memory space requirement is hence equivalent to finding the cut traversing edges such that the sum of their weights is maximal.
- This max-flow cut problem can be solved either with:
  - Mixed-Integer Linear Programming (MILP)
  - Polynomial-time algorithmic approach

\[ \max \text{sum of weights of cut edges} \]

**DAG**

- Create one node per edge
- Connect edges that cannot be on the same path

**COMPARABILITY GRAPH**

- Max-flow cut: edges with 9, 8, and 6 weights
- Max-weight independent set: edges with 9, 8, and 2 weights
EVALUATION
Evaluation

• The proposed approaches have been evaluated with both
  ➢ Synthetic workload based on randomly-generated DAG tasks
  ➢ STR2RTS benchmarks (digital signal processing applications)
  ➢ ... on 4-core and 8-core platforms

• Different node-to-core partitioning strategies have been tested
  ➢ Worst-fit, First-fit, Best-fit heuristics (e.g., based on the WCETs)
  ➢ Rank-based (kind of topological order)

### STR2RTS benchmarks

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>FFT4</td>
<td>Precise Fast Fourier Transform</td>
</tr>
<tr>
<td>B2</td>
<td>FilterBankNew</td>
<td>Multi-rate signal processing</td>
</tr>
<tr>
<td>B3</td>
<td>FMRadio</td>
<td>FM radio</td>
</tr>
<tr>
<td>B4</td>
<td>AudioBeam</td>
<td>Audio beam-forming</td>
</tr>
<tr>
<td>B5</td>
<td>Beamformer</td>
<td>Beam-forming</td>
</tr>
<tr>
<td>B6</td>
<td>CFAR</td>
<td>Constant False Alarm Detection</td>
</tr>
<tr>
<td>B7</td>
<td>FFT2</td>
<td>Fast Fourier Transform</td>
</tr>
</tbody>
</table>
Experimental Results (1)

Representative configuration with 4 processors and 5 DAG tasks
(synthetic workload)

Parameter that controls the number of per-node outgoing edges

Tend to increase the number of communications

Memory space requirement (blocks)
The lower the better

0 0.2 0.3 0.4 0.5 0.6 0.7 0.8

First-fit
Closed-form bound
Max-flow cut

Worst-fit
Closed-form bound
Max-flow cut

-24%
Experimental Results (1)

Representative configuration with **4 processors and 5 DAG tasks** (synthetic workload)

![Graph](image)

- **First-fit**
  - Closed-form bound
  - Max-flow cut

- **Worst-fit**
  - Closed-form bound
  - Max-flow cut

Parameter that controls the number of per-node outgoing edges
• Seven \((B_1, \ldots, B_7)\) benchmarks from \textbf{STR2RTS}
• \textbf{Worst-fit} partitioning based on WCETs

![Bar charts showing memory space requirement (bytes) for CPU1, CPU2, CPU3, and CPU4 across benchmarks B1 to B7, comparing Closed-form and Max-flow cut results.](image)
We proposed techniques to **bound** the maximum memory space requirement of **parallel tasks** running upon scratchpad-based multicore platforms.

Both **closed-form** and **algorithmic techniques** have been presented:

- Top-down decomposition
- Max-plus algebra (applies to fork-join tasks)
- Max-flow cut problem (applies to DAG tasks)

**Experimental results** showed that closed-form bounds are close to those provided by algorithmic techniques, especially for **real-world workload** (STR2RTS benchmarks).
Future Work

- Integrate with **timing analysis**
- ... also accounting for memory contention
- ... to develop a **holistic partitioning methodology**

Synthesize **additional** precedence constraints to **minimize** the memory space requirement while ensuring schedulability

Application to hardware accelerators
Thank you!

Alessandro Biondi
alessandro.biondi@sssup.it