Memory-bounded heuristics for parallel tree traversals

Loris Marchal (CNRS, Lyon, France)

joint work with Lionel Eyraud-Dubois, Oliver Sinnen, Frédéric Vivien
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Introduction

Problem:
- Schedule tree-shaped task graphs
- On parallel machine
- Under limited memory

Motivation:
- Multifrontal sparse matrix factorization
- Assembly/Elimination tree: application task graph is a in-tree
- Large temporary data
- Memory usage becomes a bottleneck
Tree-shaped task graphs

- In-tree of $n$ sequential tasks
- Output data of size $f_i$
- Program of size $n_i$
- Task $i$ has length $p_i$
- Input data of leaf nodes have null size
- Limited shared memory

When processing a node: input data, output data and program have to be in memory

Lessons from sequential study:
- Traversal influences memory requirement (peak)
- Postorder (=depth first) traversals naturally good for memory behavior (but not optimal)
Tree-shaped task graphs

- In-tree of \( n \) sequential tasks
- Output data of size \( f_i \)
- Program of size \( n_i \)
- Task \( i \) has length \( p_i \)
- Input data of leaf nodes have null size
- Limited shared memory

When processing a node: input data, output data and program have to be in memory

Lessons from sequential study:
- Traversal influences memory requirement (peak)
- Postorder (=depth first) traversals naturally good for memory behavior (but not optimal)
Tree-shaped task graphs

- In-tree of $n$ sequential tasks
- Output data of size $f_i$
- Program of size $n_i$
- Task $i$ has length $p_i$
- Input data of leaf nodes have null size
- Limited shared memory

When processing a node: input data, output data and program have to be in memory

Lessons from sequential study:

- Traversal influences memory requirement (peak)
- Postorder (=depth first) traversals naturally good for memory behavior (but not optimal)
Tree-shaped task graphs

- In-tree of $n$ sequential tasks
- Output data of size $f_i$
- Program of size $n_i$
- Task $i$ has length $p_i$
- Input data of leaf nodes have null size
- Limited shared memory

When processing a node: input data, output data and program have to be in memory

Lessons from sequential study:

- Traversal influences memory requirement (peak)
- Postorder (=depth first) traversals naturally good for memory behavior (but not optimal)
Tree-shaped task graphs

- In-tree of $n$ sequential tasks
- Output data of size $f_i$
- Program of size $n_i$
- Task $i$ has length $p_i$
- Input data of leaf nodes have null size
- Limited shared memory

When processing a node: input data, output data and program have to be in memory

Lessons from sequential study:
- Traversal influences memory requirement (peak)
- Postorder (=depth first) traversals naturally good for memory behavior (but not optimal)
Parallel processing of task trees

- Assume $p$ processors, shared memory
- Processing tasks in parallel $\Rightarrow$ larger memory
- Bi-objective problem: makespan and peak memory

Known results:
- Comparing to the optimal makespan and the optimal memory is difficult:
  - NP-complete
  - Not approximable with constant factor independent of $p$
How to cope with limited memory

- When processing a tree on a given machine: bounded memory
- **Objective:** Minimize makespan under this constraint
- **NB:** bounded memory $\geq$ memory for sequential processing
- **Intuition:**
  - When data sizes $<<$ mem. bound: process many tasks in parallel
  - When approaching memory bound, limit parallelism
- Rely on a (memory-friendly) sequential traversal

Several approaches:

1. Conservative: book memory to be able to get back to a sequential processing
2. Use tree structure to predict memory reuse
3. Combination of 1 and 2
How to cope with limited memory

- When processing a tree on a given machine: bounded memory
- Objective: Minimize makespan under this constraint
- NB: bounded memory $\geq$ memory for sequential processing

Intuition:
- When data sizes $<<$ mem. bound: process many tasks in parallel
- When approaching memory bound, limit parallelism
- Rely on a (memory-friendly) sequential traversal

Several approaches:
1. Conservative: book memory to be able to get back to a sequential processing
2. Use tree structure to predict memory reuse
3. Combination of 1 and 2
How to cope with limited memory

- When processing a tree on a given machine: bounded memory
- Objective: Minimize makespan under this constraint
- NB: bounded memory $\geq$ memory for sequential processing
- Intuition:
  - When data sizes $<<$ mem. bound: process many tasks in parallel
  - When approaching memory bound, limit parallelism
- Rely on a (memory-friendly) sequential traversal

Several approaches:

1. Conservative: book memory to be able to get back to a sequential processing
2. Use tree structure to predict memory reuse
3. Combination of 1 and 2
How to cope with limited memory

- When processing a tree on a given machine: bounded memory
- Objective: Minimize makespan under this constraint
- NB: bounded memory $\geq$ memory for sequential processing
- Intuition:
  - When data sizes $<<$ mem. bound: process many tasks in parallel
  - When approaching memory bound, limit parallelism
- Rely on a (memory-friendly) sequential traversal

Several approaches:

1. Conservative: book memory to be able to get back to a sequential processing
2. Use tree structure to predict memory reuse
3. Combination of 1 and 2
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:
- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

Can cope with very small memory bound

No memory reuse
**Conservative approach: task activation**

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:
- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

Can cope with very small memory bound

No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order \((book f_i + n_i)\)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

Can cope with very small memory bound

No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- 🙆 Can cope with very small memory bound
- 😞 No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:
- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a tasks complete:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Conservative approach: task activation

- Choose a sequential task order (e.g. best postorder for memory)
- While memory available, activate tasks in this order (book $f_i + n_i$)
- Schedule only activated tasks (with any priority)

When a task completes:

- Free input data and program
- Allocate as many new tasks as possible
- Then, start scheduling allocated tasks

- Can cope with very small memory bound
- No memory reuse
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Predict memory reuse: new assumptions

- Idea: reuse memory for higher level in the tree
- Book memory only when starting new leaves
- Stronger assumptions needed:
  - Reduction tree: \[ \sum_{j \in \text{Children}(i)} f_j \geq f_i \]
  - All programs have negligible sizes \( n_i = 0 \)
- For trees that do not respect these constraints, add fictitious nodes
Predict memory reuse: new assumptions

- Idea: reuse memory for higher level in the tree
- Book memory only when starting new leaves
- Stronger assumptions needed:
  - Reduction tree: \[ \sum_{j \in \text{Children}(i)} f_j \geq f_i \]
  - All programs have negligible sizes \( (n_i = 0) \)
- For trees that do not respect these constraints, add fictitious nodes
Predict memory reuse: new assumptions

- Idea: reuse memory for higher level in the tree
- Book memory only when starting new leaves
- Stronger assumptions needed:
  - Reduction tree: $\sum_{j \in \text{Children}(i)} f_j \geq f_i$
  - All programs have negligible sizes ($n_i = 0$)
- For trees that do not respect these constraints, add fictitious nodes
Predict memory reuse: first attempt

- Follow sequential postorder with seq. peak $M_{seq} \leq M_{bound}$
- List scheduling + constraints:
  - When starting a leaf $c$, check that $M_{used} + f_c \leq M_{bound}$
  - Otherwise, wait for some memory to be freed

Theorem (The peak memory is not larger than $2M_{bound}$).

- Completes the whole tree (because sequential order valid)
- When starting a new leaf:
  
  $M_{used} = \text{Input}\_\text{inner} + \text{Output}\_\text{inner} + \text{Output}\_\text{leaves} + \text{InputIdle} \leq M_{bound}$

- Later, when processing inner nodes, without starting new leaves:
  
  sum of all inputs $\leq \text{Input}\_\text{inner} + \text{InputIdle} + \text{Output}\_\text{leaves} \leq M_{bound}$
  
  sum of all outputs $\leq$ sum of all inputs

  total memory $\leq 2M_{bound}$
Predict memory reuse: first attempt

- Follow sequential postorder with seq. peak $M_{seq} \leq M_{bound}$
- List scheduling + constraints:
  - When starting a leaf $c$, check that $M_{used} + f_c \leq M_{bound}$
  - Otherwise, wait for some memory to be freed

Theorem (The peak memory is not larger than $2M_{bound}$).

- Completes the whole tree (because sequential order valid)
- When starting a new leaf:
  
  
  $M_{used} = Input_{INNER} + Output_{INNER} + Output_{LEAVES} + InputIdle \leq M_{bound}$

- Later, when processing inner nodes, without starting new leaves:
  - sum of all inputs: $\leq Input_{INNER} + InputIdle + Output_{LEAVES} \leq M_{bound}$
  - sum of all outputs $\leq$ sum of all inputs
  - total memory $\leq 2M_{bound}$
Predict memory reuse: first attempt

- Follow sequential postorder with seq. peak $M_{seq} \leq M_{bound}$
- List scheduling + constraints:
  - When starting a leaf $c$, check that $M_{used} + f_c \leq M_{bound}$
  - Otherwise, wait for some memory to be freed

Theorem (The peak memory is not larger than $2M_{bound}$).

- Completes the whole tree (because sequential order valid)
  - When starting a new leaf:
    \[
    M_{used} = \text{Input}_{INNER} + \text{Output}_{INNER} + \text{Output}_{LEAVES} + \text{InputIdle} \leq M_{bound}
    \]
  - Later, when processing inner nodes, without starting new leaves:
    - sum of all inputs: $\leq \text{Input}_{INNER} + \text{InputIdle} + \text{Output}_{LEAVES} \leq M_{bound}$
    - sum of all outputs $\leq$ sum of all inputs
    - total memory $\leq 2M_{bound}$
Predict memory reuse: first attempt

**Theorem** (The peak memory is not larger than $2M_{\text{bound}}$).
- Completes the whole tree (because sequential order valid)
- When starting a new leaf:
  \[
  M_{\text{used}} = \text{Input}_{\text{INNER}} + \text{Output}_{\text{INNER}} + \text{Output}_{\text{LEAVES}} + \text{InputIdle} \leq M_{\text{bound}}
  \]
- Later, when processing inner nodes, without starting new leaves:
  - sum of all inputs: $\leq \text{Input}_{\text{INNER}} + \text{InputIdle} + \text{Output}_{\text{LEAVES}} \leq M_{\text{bound}}$
  - sum of all outputs $\leq$ sum of all inputs
  - total memory $\leq 2M_{\text{bound}}$
Predict memory reuse: first attempt

Theorem (The peak memory is not larger than $2M_{\text{bound}}$).

- Completes the whole tree (because sequential order valid)
- When starting a new leaf:

  $$M_{\text{used}} = Input_{\text{INNER}} + Output_{\text{INNER}} + Output_{\text{LEAVES}} + Input_{\text{idle}} \leq M_{\text{bound}}$$

- Later, when processing inner nodes, without starting new leaves:
  - sum of all inputs: $\leq Input_{\text{INNER}} + Input_{\text{idle}} + Output_{\text{LEAVES}} \leq M_{\text{bound}}$
  - sum of all outputs $\leq$ sum of all inputs
  - total memory $\leq 2M_{\text{bound}}$
Predict memory reuse: second attempt

- Previous heuristic: ⇒ Simple algorithm, $2M_{\text{bound}}$ guarantee
- Can we get a better guarantee while predicting memory reuse?
- Book memory for parent nodes, ensure they can be processed later:
  \[
  \text{Contrib}[j] = \min \left( inputs(j), f_i - \sum_{j' \in \text{Children}(i)} \text{Contrib}[j'] \right)
  \]
  \[
  j' \in \text{Children}(i) \quad \text{and} \quad \text{PO}(j') > \text{PO}(j)
  \]
- Test for memory (booked+used) when starting a leaf
- When to book memory:
  - At inner node completion,
  - When starting a leaf.

- 😊 Never exceeds a given memory $M_{\text{bound}}$
- 😞 Extra memory weights to get a reduction tree
Predict memory reuse: second attempt

- Previous heuristic: \( \Rightarrow \) Simple algorithm, \( 2M_{\text{bound}} \) guarantee
- Can we get a better guarantee while predicting memory reuse?
- Book memory for parent nodes, ensure they can be processed later:
  \[
  \text{Contrib}[j] = \min \left( \text{inputs}(j), f_i - \sum_{j' \in \text{Children}(i)} \text{Contrib}[j'] \right)
  \]
- Test for memory (booked + used) when starting a leaf
- When to book memory:
  - At inner node completion,
  - When starting a leaf.

- \( \smiley \) Never exceeds a given memory \( M_{\text{bound}} \)
- \( \frown \) Extra memory weights to get a reduction tree
Predict memory reuse: second attempt

- Previous heuristic: ⇒ Simple algorithm, $2M_{bound}$ guarantee
- Can we get a better guarantee while predicting memory reuse?
- Book memory for parent nodes, ensure they can be processed later:

$$Contrib[j] = \min \left( inputs(j), f_i - \sum_{j' \in \text{Children}(i)} \sum_{PO(j') > PO(j)} Contrib[j'] \right)$$

- Test for memory (booked+used) when starting a leaf
- When to book memory:
  - At inner node completion,
  - When starting a leaf.

- Always never exceeds a given memory $M_{bound}$
- Extra memory weights to get a reduction tree
Predict memory reuse: second attempt

- Previous heuristic: ⇒ Simple algorithm, $2M_{\text{bound}}$ guarantee
- Can we get a better guarantee while predicting memory reuse?

- Book memory for parent nodes, ensure they can be processed later:

  $$\text{Contrib}[j] = \min \left( \left( \text{inputs}(j), f_i - \sum_{j' \in \text{Children}(i), j' \in \text{Children}(i)} \text{Contrib}[j'] \right) \right)$$

- Test for memory (booked+used) when starting a leaf

- When to book memory:
  - At inner node completion,
  - When starting a leaf.

- ☑️ Never exceeds a given memory $M_{\text{bound}}$
- ☹️ Extra memory weights to get a reduction tree
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Mixing both approaches

- Idea: refine the activation using memory booking
- Book exactly what is needed for each activated subtree
- \textit{ParaPeak}(i): maximal amount of memory needed to process subtree rooted in \( i \) in parallel

\[
\text{ParaPeak}(i) = \begin{cases} 
  f_i & \text{if } i \text{ has already been processed} \\
  f_i + n_i & \text{if } i \text{ is a leaf} \\
  \max \left( f_i + n_i + \sum_{j \in \text{Children}(i)} f_j, \sum_{j \in \text{Children}(i)} \text{ParaPeak}(j) \right) & \text{otherwise.}
\end{cases}
\]

- Modify the activation algorithm such that:

\[
\text{ActivatedMemory} = \sum_{i \text{ root of a maximal activated subtree}} \text{ParaPeak}(i).
\]
Changes for a refined activation

- When testing/allocating a new node:
- When completing a node $i$:
  - Set $ParaPeak \leftarrow f_i$
  - Recompute $ParaPeak$ for its parent, if it changes, continue with grand-parent, ...
  - Until we reach a node not activated yet.

- Compute its $ParaPeak$ based on its children’s $ParaPeak$.
- Compute the new overall memory needed. If less than the bound, proceed.

- 😊 Accurate memory prediction (more nodes activated)
- 😊 Can cope with the minimum (postorder) memory
- 😞 Upon node completion, (many) updates of $ParaPeak$
Outline

Introduction

Conservative approach: task activation

Predictive approach: memory reuse

Mixing both approaches

Preliminary results
Simulation testbed

- **Set 1: Synthetic regular (heterogeneous) trees**
  - 480 fully balanced trees
  - $3 \leq \text{height} \leq 8$
  - $2 \leq \text{degree} \leq 5$
  - $1 \leq \text{file and node sizes} \leq 20$
  - $2 \leq \text{processing times} \leq 20$

- **Set 2: Assembly trees of sparse matrices**
  - 76 matrices from University of Florida Sparse Collection
  - Metis and AMD ordering
  - 1, 2, 4, or 16 relaxed amalgamation per node
  - 608 trees with:
    - number of nodes: 2,000 to 1,000,000
    - depth: 12 to 70,000
    - maximum degree: 2 to 175,000

- 8 processors
Regular trees: makespan vs. memory bound

Activation+Reuse outperforms both Activation and Reuse
Assembly trees: makespan vs. memory bound

Larger gap between different strategies
Assembly trees: memory use

Fraction of the available memory really used

Normalized amount of limited memory

heuristic
- BasicReuse
- Activation
- Activation+Reuse
- RefinedReuse
Assembly trees: scheduling time

![Graph showing scheduling time for different heuristics.]

- **heuristic**
  - BasicReuse
  - Activation
  - Activation+Reuse
  - RefinedReuse

- **Scheduling time in seconds (logarithmic scale)**
- **Number of nodes (logarithmic scale)**
Conclusion

- Parallel processing of task trees with guaranteed memory
- To minimize makespan:
  - Use as much memory as possible
  - Use activation + memory reuse

Perspectives:
- Reduce scheduling time for runtime execution
- Test other sequential orderings, choose among them at runtime
- Consider parallel (malleable?) tasks
- Move to distributed memory