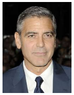
A performance model to execute workflows on high-bandwidth memory architectures



Anne Benoit[1,3] Swann Perarnau[4]
Loïc Pottier[1] Yves Robert[1,2]

[1]ENS Lyon & Inria, France
[2]University of Tennessee Knoxville, USA
[3]Georgia Institute of Technology, USA
[4]Argonne National Laboratory, USA

Yves.Robert@inria.fr

ICL – April 27, 2018

Motivation

Focus on KNL architecture.

- Cache mode: MCDRAM used by hardware as last-level cache
- Flat mode: MCDRAM manually managed by programmers
- Hybrid mode: mixes both previous modes

- Cache mode easy to use (nothing to do)
- Flat mode?

Motivation

Focus on KNL architecture.

- Cache mode: MCDRAM used by hardware as last-level cache
- **Flat mode: MCDRAM manually managed by programmers**
- Hybrid mode: mixes both previous modes

- Cache mode easy to use (nothing to do)
- Flat mode?

no performance model available! (yet)

Problem

Given:

- a scientific workflow, represented as a DAG $\rightarrow G = (V, E)$
- a deep-memory architecture with P cores (a.k.a. processors)

Goal

Find a scheduling and memory mapping that minimizes the execution time of the workflow (a.k.a. makespan)

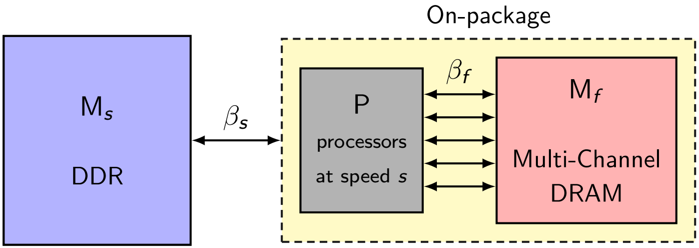
Outline

- Model and complexity
- Heuristics
- Simulation results
- Experimental results on a 1D stencil
- Conclusion

Model

Architecture

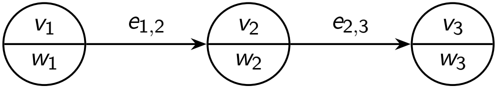
- Large slow-bandwidth memory M_s with bandwidth β_s
- Small high-bandwidth memory M_f with bandwidth β_f
- P identical processors processing at speed s



Model

Application

- Acyclic graph $G = (V, E)$
- Node set $V = \{v_1, \ldots, v_n\}$: **sequential** computational tasks
- Edges $E \subseteq V^2$ are dependences between tasks
- Each node v_i has a number of computing operations: w_i
- Each edge $(v_i, v_j) \in E$ has a number of data blocks: $e_{i,j}$



How to compute the execution time?

Resources (bandwidth and memory) are concurrently used ☹

$\rightarrow$ partition execution into *events* $(t_1, \dots, t_k, \dots, t_{2n})$

Chunk

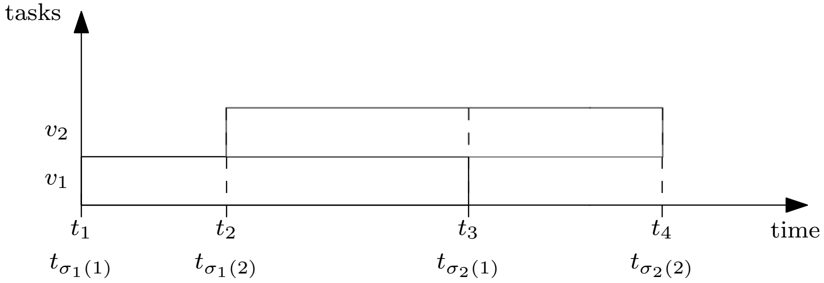
A *chunk* is a period of time between two consecutive events. The chunk k is defined as $t_{k+1} - t_k$.

$\rightarrow$ during a *chunk*,both bandwidths and memory occupations remain constant ☺

Events

To express all scheduling constraints, we define two events:

- $\sigma_1(i) \rightarrow$ time-step when v_i starts
- $\sigma_2(i) \rightarrow$ time-step when v_i ends
- $2n$ events at most (recall, $n = |V|$)



Events

- $w_i^{(k)}$ number of operations executed by task v_i during chunk k
- $\beta_f^{(k)}$ fast bandwidth available for each task during chunk k (shared equally among executing tasks)
  – for slow bandwidth, $\beta_s^{(k)}$
- $\mathrm{S}_f^{(k)}$ number of blocs allocated into the fast memory M_f during chunk k

Scheduling constraints

Precedence constraint	$\forall (v_i, v_j) \in E,\ t_{\sigma_2(i)} \leq t_{\sigma_1(j)}$
Memory size constraint	$\forall 1 \leq k \leq 2n-1,\ \mathsf{S}_f^{(k)} \leq \mathsf{S}_f$
Processor constraint	$\left\| \left\{ v_i \mid t_{\sigma_1(i)} \leq t < t_{\sigma_2(i)} \right\} \right\| \leq \mathsf{P}$
Work constraint	$\forall i, \quad \sum_{k=1}^{2n-1} w_i^{(k)} = w_i$

Problem

At schedule time, for a task v_i, choose for each successor v_j:

- $e^f_{i,j}$ number of data blocks in fast memory M_f
- $e^s_{i,j}$ number of data blocks in slow memory M_s
  $(e^f_{i,j} + e^s_{i,j} = e_{i,j})$

(And decide which task is scheduled on which available processor)

Problem

At schedule time, for a task v_i, choose for each successor v_j:

- $e^f_{i,j}$ number of data blocks in fast memory M_f
- $e^s_{i,j}$ number of data blocks in slow memory M_s
  $(e^f_{i,j} + e^s_{i,j} = e_{i,j})$

(And decide which task is scheduled on which available processor)

MemDag

Given an acyclic graph G and the platform, find a memory mapping $\mathcal{X} = \{e^f_{i,j}\}_{(v_i,v_j)\in E}$ and a schedule $\{t_{\sigma_1(i)}, t_{\sigma_2(i)}\}_{v_i\in V}$ satisfying all constraints and minimizing

$$\max_{v_i\in V} \; t_{\sigma_2(i)}$$

Execution time

The total number of blocks read/written from fast memory for v_i:

- $in_i^f = \sum_{v_j \in \text{pred}(v_i)} e_{j,i}^f$ and $out_i^f = \sum_{v_j \in \text{succ}(v_i)} e_{i,j}^f$

Execution time

The total number of blocks read/written from fast memory for v_i:

- $in_i^f = \sum_{v_j \in \mathrm{pred}(v_i)} e_{j,i}^f$ and $out_i^f = \sum_{v_j \in \mathrm{succ}(v_i)} e_{i,j}^f$

Computations $$\frac{w_i^{(k)}}{s} \le t_{k+1} - t_k$$

Fast communications $$\frac{w_i^{(k)}}{w_i} \times \frac{in_i^f + out_i^f}{\beta_f^{(k)}} \le t_{k+1} - t_k$$

Slow communications $$\frac{w_i^{(k)}}{w_i} \times \frac{in_i^s + out_i^s}{\beta_s^{(k)}} \le t_{k+1} - t_k$$

Execution time (1/2)

Work done by v_i during chunk k:

$$w_i^{(k)} = (t_{k+1} - t_k) \min\left(s, \frac{\beta_f^{(k)} w_i}{in_i^f + out_i^f}, \frac{\beta_s^{(k)} w_i}{in_i^s + out_i^s}\right)$$

Pessimistic formulation:

$$\frac{w_i^{(k)}}{s} + \frac{w_i^{(k)}}{w_i} \cdot \frac{in_i^f + out_i^f}{\beta_f^{(k)}} + \frac{w_i^{(k)}}{w_i} \cdot \frac{in_i^s + out_i^s}{\beta_s^{(k)}} \leq t_{k+1} - t_k$$

Execution time (2/2)

Next time-step

$$E_i^{(k)} = t_k + \frac{w_i - \sum_{k'=\sigma_1(i)}^{k-1} w_i^{(k')}}{\min\left(s, \frac{\beta_f^{(k)} w_i}{in_i^f + out_i^f}, \frac{\beta_s^{(k)} w_i}{in_i^s + out_i^s}\right)}.$$

$$t_{k+1} = \min_{v_i \in V} E_i^{(k)}$$

Completion

$$\sum_{k=1}^{2n-1} w_i^{(k)} = w_i, \quad \sum_{k=1}^{2n-1} in_i^{f(k)} = in_i^f, \quad \dots$$

Makespan

$$\mathcal{T} = \max_{v_i \in V} \; t_{\sigma_2(i)}$$

Execution time (2/2)

Next time-step

$$w_i - \sum^{k-1} w_i^{(k')}$$

Candid complaint

And you said you had a "simplified model" ☹☹☹

$v_i \in V$

Completion

Complexity

- MemDag is NP-Complete in the strong sense (of course)
- Complexity unknown for linear chains

Linear chains

$$\xrightarrow{e_{0,1}} v_1 \xrightarrow{e_{1,2}} v_2 \rightarrow \cdots \rightarrow v_i \xrightarrow{e_{i,i+1}} v_{i+1} \rightarrow \cdots \rightarrow v_n \xrightarrow{e_{n,n+1}}$$

.

MINIMIZE $\sum_{i=1}^{n} m_i$

SUBJECT TO
$$\left\{\begin{array}{ll} e_{i,i+1} = e^s_{i,i+1} + e^f_{i,i+1} & \text{for } 0 \leq i \leq n \\ \frac{w_i}{s} \leq m_i & \text{for } 1 \leq i \leq n \\ \frac{e^s_{i-1,i} + e^s_{i,i+1}}{\beta_s} \leq m_i & \text{for } 1 \leq i \leq n \\ \frac{e^f_{i-1,i} + e^f_{i,i+1}}{\beta_f} \leq m_i & \text{for } 1 \leq i \leq n \\ e^f_{i-1,i} + e^f_{i,i+1} \leq S_f & \text{for } 1 \leq i \leq n \end{array}\right.$$

Heuristics

NP-Complete problem $\rightarrow$ polynomial-time heuristics.
Need to:

- Decide which task executes first (processor allocation)
- Decide in which memory a given task will write its data blocks (memory mapping)

Metrics

Two metrics used to sort tasks:

Critical Path of v_i

$$CP_i = \max\left(\frac{w_i}{s}, \frac{in_i + out_i}{\beta_s}\right) + \max_{j \in \mathrm{succ}(v_i)} CP_j.$$

Gain

The *gain* of a subgraph G_i rooted at v_i is defined as:

$$gain(G_i) = \frac{Bl_f(G_i)}{Bl_s(G_i)},$$

where $Bl_s(G_i)$ is the makespan of G_i when using no fast memory and $Bl_f(G_i)$ is the makespan when using an infinite fast memory.

Main algorithm

Main algorithm based list scheduling:

- Ready tasks sorted according to scheduling policy
- For each scheduled task v_i at that step:
    - For each successor of v_i, a memory mapping decides the value of each $e^f_{i,j}$
- And so on, until all tasks have been scheduled

Processor scheduling policies

Let $L^{(k)}$ be the list of ready tasks to be executed at chunk k.

- CP sorts $L^{(k)}$ according to their critical paths (non-increasing order of CP_i with $v_i \in L^{(k)}$)
- GG sorts $L^{(k)}$ according to their potential gains (non-decreasing order of $gain(G_i)$ with $v_i \in L^{(k)}$)

Memory mapping policies

A memory mapping must decide in which memory a task will write its data blocks:

- MemCP greedily gives fast memory blocks to each successor of v_i according to their critical paths
- MemGG greedily gives fast memory blocks to each successor of v_i according to their gains
- MemFair greedily gives data blocks in fast memory to the tasks, according to their amount of computations, but accounting for other successors

Two memory mappings

Algorithm 1: Heuristic MemCP

```
1 procedure MemCP ($v_i$)
  begin
2     Let $U$ be the set of $v_i$'s successors ordered by $CP_i$ ;
3     $\mathcal{X} \leftarrow \emptyset$ ;
4     foreach $j \in U$ do
5         $e^f_{i,j} \leftarrow \min\left(\mathsf{S}_f - \mathsf{S}^{(k)}_f, e_{i,j}\right)$;
6         $\mathcal{X} \leftarrow \mathcal{X} \cup \{e^f_{i,j}\}$ ;
7         $\mathsf{S}^{(k)}_f \leftarrow \mathsf{S}^{(k)}_f + e^f_{i,j}$ ;
8     return $\mathcal{X}$ ;
```

Algorithm 2: Heuristic MemFair

```
1 procedure MemFair ($v_i$)
  begin
2     Let $U$ be the set of $v_i$'s successors ordered by $w_i$ ;
3     $\mathcal{X} \leftarrow \emptyset$ ;
4     foreach $j \in U$ do
5         $e^f_{i,j} \leftarrow \min\left(\left\lfloor \frac{\mathsf{S}_f - \mathsf{S}^{(k)}_f}{|\mathrm{succ}(v_i)|} \right\rfloor, e_{i,j}\right)$;
6         $\mathcal{X} \leftarrow \mathcal{X} \cup \{e^f_{i,j}\}$ ;
7         $\mathsf{S}^{(k)}_f \leftarrow \mathsf{S}^{(k)}_f + e^f_{i,j}$ ;
8     return $\mathcal{X}$ ;
```

Baseline heuristics

- CP +NoFast: no fast memory is available
- CP +InfFast: infinite fast memory (finite bandwidth)
- CP +CcMode: imitation of the KNL cache mode behavior (fast memory divided into P slices, each task can use at most $\frac{S_f}{P}$ data blocks)

Simulations results

Random graphs using Standard Tasks Graphs sets of 50 and 100 nodes (180 graphs per size)

- Two data sets for each size:
  - Sparse: the 20 graphs that exhibit the lower densities (only sparse results in this talk)
  - Dense: the 20 graphs that exhibit the higher densities

Parameters based on KNL architecture:

- $\beta_s = 90$ GB/s and $\beta_f = 450$ GB/s (five times faster)
- Processor speed $s = 1.4$ GHz
- Fast memory of size $S_f = 16$ GB (infinitely large slow memory)

Simulations results

Generating computation weights w_i and communication costs $e_{i,j}$:

- w_i uniformly drawn in $[w_i^{\min} = 10^4, w_i^{\max} = 10^6]$ flops
- Computations to Communication Ratio:

$$\mathrm{CCR} = \frac{w_i}{s} \Big/ \frac{e_{i,j}}{\beta_s}.$$

- $e_{i,j}$ is uniformly and randomly pick in

$$\left[\frac{w_i^{\min} \times \beta_s}{s \times \mathrm{CCR}}, \frac{w_i^{\max} \times \beta_s}{s \times \mathrm{CCR}} \right]$$

Impact of the number of processors

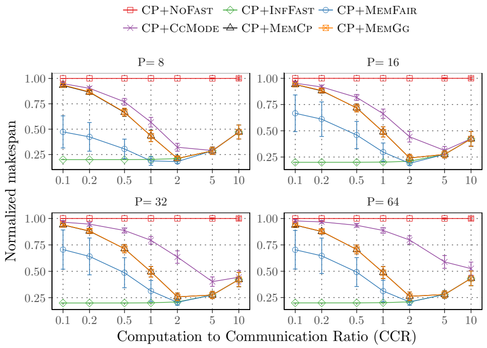


Figure: Impact of the number of processors with 50 nodes and $S_f = 1$GB fast memory for CP scheduling heuristic.

Heuristics very efficient (gain around 50%)
CP +MemFair even outperforms CP +CcMode!

Impact of fast memory size

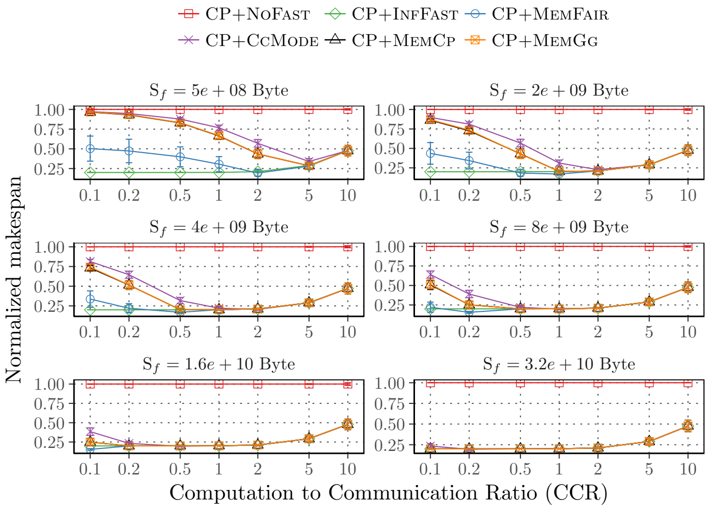


Figure: Impact of fast memory size with 50 nodes and 8 processors.

Experimental 1D stencil

Comparison of simulations and real experiments
$\rightarrow$ assess model accuracy

- 1D stencil kernel running on a KNL
- Pipelined approach: for a given iteration in parallel:
    - prefetch future tiles in fast memory
    - compute on tiles already prefetched
    - flush computed tiles back into slow memory

Algorithm 3: 1D Gauss-Seidel algorithm

```
1 procedure 1D-GS(array) begin
2     for $t = 1$ to ... do
3         for $i = 1$ to ... do
4             $\text{Tile}_i^t \leftarrow \text{Gauss-Seidel}\left(\text{Tile}_{i-1}^t, \text{Tile}_i^{t-1}, \text{Tile}_{i+1}^{t-1}\right)$;
```

Task graph

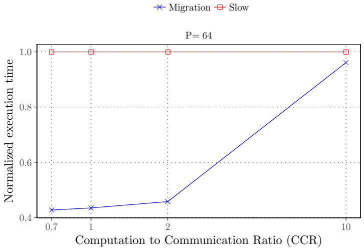
But our model does allow (yet ☺) data transfers:

- Update of each tile is decomposed into three sequential tasks:
  - R_i^t transfers the tile from slow memory to fast memory
  - C_i^t computes the tile in fast memory
  - W_i^t writes the updated tile back into slow memory

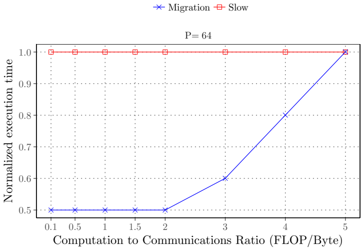
$$\begin{tikzcd}
R_0^t \arrow[d] & \cdots & R_{i-1}^t \arrow[r, "0"] \arrow[d] & R_i^t \arrow[r, "0"] \arrow[d] & R_{i+1}^t \arrow[d] & \cdots & R_n^t \arrow[d] \\
C_0^t \arrow[d] & \cdots & C_{i-1}^t \arrow[r, "0"] \arrow[d] & C_i^t \arrow[r, "0"] \arrow[d] & C_{i+1}^t \arrow[d] & \cdots & C_n^t \arrow[d] \\
W_0^t \arrow[d, "m"] & \cdots & W_{i-1}^t \arrow[r, "0"] \arrow[d, "m"'] & W_i^t \arrow[r, "0"] \arrow[d, "m"] \arrow[dl, "m"] & W_{i+1}^t \arrow[d, "m"] \arrow[dl, "m"] & \cdots & W_n^t \arrow[d, "m"] \\
R_0^{t+1} \arrow[d] & \cdots & R_{i-1}^{t+1} \arrow[r, "0"] \arrow[d] & R_i^{t+1} \arrow[r, "0"] \arrow[d] & R_{i+1}^{t+1} \arrow[d] & \cdots & R_n^{t+1} \arrow[d] \\
C_0^{t+1} \arrow[d] & \cdots & C_{i-1}^{t+1} \arrow[r, "0"] \arrow[d] & C_i^{t+1} \arrow[r, "0"] \arrow[d] & C_{i+1}^{t+1} \arrow[d] & \cdots & C_n^{t+1} \arrow[d] \\
W_0^{t+1} & \cdots & W_{i-1}^{t+1} \arrow[r, "0"] & W_i^{t+1} \arrow[r, "0"] & W_{i+1}^{t+1} & \cdots & W_n^{t+1}
\end{tikzcd}$$

Figure: t is the iteration index, i is the tile index, and m is the size of one tile.

Experimental results



(a) KNL



(b) Model

Figure: Performance of a 1D stencil with 64 threads.

Good concordance between experiments and simulations!
Best performance when access to slow memory is the bottleneck.

Conclusion

Contributions

- Detailed performance model for scheduling workflows on dual memory-systems
- Efficient polynomial heuristics
- Simulations and real experiments on a 1D stencil

Future work

- Extend simulations to other workflow graphs (fork-joins, etc)
- Allow prefetching into fast memory
- Additional experiments with more complicated workflows