Analysis of Dynamic Scheduling Strategies for Matrix Multiplication on Heterogeneous Platforms

Olivier Beaumont (Inria) and Loris Marchal (CNRS)
Outline

Introduction

Outer Product

Matrix Product

Extensions

Conclusion and Perspectives
Scheduling in Large Scale Systems

- Scheduling and resource allocation problems are hard due to
  - Failures
  - Non-determinism of execution/transfer time
  - Heterogeneity

- Deciding in advance:
  - Where to place tasks
  - When/In which order executing them
can make the system very slow

- Makes the interest of classical static strategies questionable
Introduction

Solutions to cope with Uncertainties

▶ On the theoretical side: robust scheduling
  ▶ Given probability distribution of execution/transfer time
  ▶ Find (allocation/schedule) that minimize expected makespan
  ▶ Worst case is also interesting and not trivial
  ▶ Bad point: extremely difficult (and limited to small instances)

▶ On the practical side:
  ▶ Fault Tolerance: checkpointing, replication
  ▶ Scheduling and Load Balancing:
    Mostly dynamic, runtime, demand driven strategies
    (e.g. Hadoop, PaRSEC, StarSs, KAALI, StarPU, Swift)
  ▶ Dynamic (runtime) decisions based on
    ▶ state of the resources
    ▶ estimations of execution/transfer times
    ▶ possibly some static priority mechanism between ready tasks
  ▶ Basic strategies but efficient in practice
On the application side

- Dynamic schedulers perfectly suited for independent tasks:
  - MapReduce, Divisible Load Tasks
  - Independent tasks are what dynamic schedulers see

- But used more and more for applications with dependencies...

- Lessons learned from dynamic task-based runtime systems:
  - Placing tasks close to the actual data location is crucial
  - Placing tasks on well adapted resources is crucial
  - Deciding task order is not so crucial, but avoid idle time

- Still, many open questions!
  - Dynamic Strategies:
    - What is the impact of deciding at runtime (myopic vision) ?
  - Static Strategies:
    - What is the impact of bad estimations (astigmatic vision) ?
  - What is the worse ?
What is our goal is this talk?

- Propose analytical models for basic dynamic strategies
- Analyze dynamic strategies to understand:
  - What makes them efficient?
  - What can be done to improve them?
  - *In particular in order to tune their parameters*
- For basic applications first
  - Linear algebra without task dependencies
  - *But in presence of tasks sharing input data*
- Today: outer product (and matrix multiplication)
Outline

Introduction

Outer Product

Matrix Product

Extensions

Conclusion and Perspectives
Outer Product

Basic quadratic complexity operation

- 2 vectors of $N$ blocks: $a$ and $b$
- Output $M = ab^T$, i.e. all $M_{i,j} = a_i \cdot b_j$ values, $1 \leq i, j \leq N$
- $N^2$ basic operations to be done (could be scalar operations).

If $P_k$ is responsible for computing the red $M_{i,j}$ values

- it needs to store the corresponding red values of $a$ and $b$

Communication Cost 6 vs 15 and processing = 9 in both cases

Less communications, same processing
Load-balancing is easy ensured by demand-driven strategy

Let $s_k$ denote the relative speed of $P_k$

Load balancing, work for $P_k$: $w_k = x_k \times y_k = s_k$

Amount of data: $V_k = x_k + y_k$

Problem: partition the square $[0; 1] \times [0; 1]$ so that

$x_k \times y_k = s_k$

$\sum_k x_k + y_k$ is minimized

Well known combinatorial problem

NP Complete

Lower bound: $2 \sum \sqrt{s_k}$

7/4-approximation algorithm
Randomized Dynamic Strategies

Basic strategies: When requested for a new task,

- Send tasks in lexicographic \((i, j)\) order (SORTED)
- Send a random (fresh) task (RANDOM)

Expected: large amount of communications, due to replication
**Randomized StarPU-like Dynamic Strategy**

Simple data-aware strategy **Dynamic**: Idea: favor tasks for which processors already hold some data.

1. When $P_k$ requests a task, send a new $a_i$ and a new $b_j$ to $P_k$
2. Allocate all available tasks $a_i \times b_{j'}$ (for $b_{j'}$ already on $P_k$)
3. Allocate all available tasks $a_{i'} \times b_j$ (for $a_{i'}$ already on $P_k$)
Randomized StarPU-like Dynamic Strategy

Simple data-aware strategy **DYNAMIC**: 
Idea: favor tasks for which processors already hold some data.

1. When $P_k$ requests a task, send a new $a_i$ and a new $b_j$ to $P_k$
2. Allocate all available tasks $a_i \times b_{j'}$ (for $b_{j'}$ already on $P_k$)
3. Allocate all available tasks $a_{i'} \times b_j$ (for $a_{i'}$ already on $P_k$)
Randomized StarPU-like Dynamic Strategy

Simple data-aware strategy **Dynamic**:

Idea: favor tasks for which processors already hold some data.

1. When $P_k$ requests a task, send a new $a_i$ and a new $b_j$ to $P_k$
2. Allocate all available tasks $a_i \times b_j'$ (for $b_j'$ already on $P_k$)
3. Allocate all available tasks $a_i' \times b_j$ (for $a_i'$ already on $P_k$)
Basic Dynamic Strategies – Comparison

- 100 blocks, 10,000 tasks
- Normalized to lower bound $2 \sum \sqrt{s_k}$, possibly too optimistic
- Lexicographic order (SORTED) is the worst: rows tend to be known by all processors
Shortcomings of Dynamic

- **Dynamic**
  - When $P_k$ requests a task, send new $a_i$ and new $b_j$ to $P_k$
  - Allocate all available tasks that can be processed on $P_k$

- Limitation:
  - When very few tasks remain to be processed
  - Sending new $a_i$ and new $b_j$ may not allow $P_k$ to do fresh tasks
  - → useless communications (before doing unprocessed tasks)

- New strategy: **Dynamic2Phases**
  - Start with **Dynamic**
  - Switch to **Random** at some point:
    - Allocate any unprocessed task $T_{i,j}$
    - Send (if necessary) the 2 corresponding $a_i$ and $b_j$ data blocks

- Question: When to switch between both strategies?
**Dynamic2Phases: Threshold**

- $P = 20$ processors, 100 blocks, 10,000 tasks
- Allows to reduce communications even more
- Optimal threshold: a few percent
- How to determine this threshold?
Assume that the size $N$ of both vectors is large

Consider a continuous dynamic system with close behavior

Describe the continuous system using Ordinary Differential Equations

Ratio $x = y/N$ of elements of $a$ (and $b$) on $P_k$ at $t_k(x)$

Basic step: when this ratio goes from $x$ to $x + \delta x = y/N + \ell/N$

In $\square$: all tasks processed (by $P_k$ or other processors)
Dynamic2Phases: Analysis (II)

- $y = xN$

- In $g_k(x)$ is the fraction of unprocessed tasks (assumed uniformly distributed).

- Time for $P_k$ to compute the red tasks:

  $$\frac{2x \delta x g_k(x) N^2}{s_k} = t_k(x + \delta x) - t_k(x)$$

- Number of tasks from computed by other processors during this step: $(t_k(x + \delta x) - t_k(x)) \sum_{i \neq k} s_i$

- Evolution of $g_k(x)$:

  $$g_k(x + \delta x) - g_k(x) = g_k(x) \delta x \frac{-2x \alpha_k}{1 - x^2}$$

  where

  $$\alpha_k = \frac{\sum_{i \neq k} s_i}{s_k}$$

  $$\Rightarrow g_k(x) = (1 - x^2)^{\alpha_k}$$
Next steps:

- determine $h_k(x)$, the number of the tasks
  - in the grey area
  - but not processed by $P_k$
  - $h_k$ is solution of a simple ODE.

- We can now compute the evolution of $t_k(x)$
  - since $(xN)^2 = h_k(x) + t_k(x)s_k$.

- Observations:
  - Let us switch when $P_k$ receives $\sqrt{\beta}$ times what it should have
  - At first order, $t_k(x)$ does not depend on $k$
  - All processors make progress at their relative speed
  - $\implies \beta$ can be used as a global threshold
For a given threshold value of $\beta$, we can estimate:
- the time taken by DYNAMIC before the switch
- the time taken by RANDOM after the switch
- And determine the "optimal" value of $\beta$

It works very well in practice!

Comparison discrete simulation vs. continuous analysis:

![Graph showing comparison between Analysis of Dynamic Scheduling Strategies and Analysis2Phases]
Simulations

Comparison with previous heuristics:

\[ \beta \] has a very small deviation with the speed distribution

- \( N \) and \( P \) being fixed,
- even with exponentially distributed heterogeneous speeds
- 100 tries, \( (\beta_{\text{opt}}^{\text{max}} - \beta_{\text{opt}}^{\text{min}})/\beta_{\text{opt}}^{\text{homogeneous}} < 5\% \).

Runtime estimation of \( \beta \): use homogeneous speeds
Outline

Introduction

Outer Product

Matrix Product

Extensions

Conclusion and Perspectives
Matrix-Product  \( C = AB \)

- Very similar problem: from 2D to 3D
- Basic computation task:  \( C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j} \)
- Send elements of \( A \) and \( B \), gather elements from \( C \)
- \( A, B \) and \( C \) need to be replicated
- Minimize communication amount
Matrix-Product: Simple Data-Aware Strategy

- Adapt the previous heuristic:
  - $P_k$ knows $xN \times xN$ values of $A$, $B$ and $C$
  - When $P_k$ requests some work
    - Send $2x - 1$ data blocks of $A$, $B$, $C$
    - Allocate all tasks available with these new data
Matrix-Product: Simple Data-Aware Strategy

- Adapt the previous heuristic:
  - $P_k$ knows $xN \times xN$ values of $A$, $B$ and $C$
  - When $P_k$ requests some work
  - Send $2x - 1$ data blocks of $A$, $B$, $C$
  - Allocate all tasks available with these new data
Matrix-Product: Simple Data-Aware Strategy

- Adapt the previous heuristic:
  - $P_k$ knows $xN \times xN$ values of $A$, $B$ and $C$
  - When $P_k$ requests some work
  - Send $2x - 1$ data blocks of $A$, $B$, $C$
  - Allocate all tasks available with these new data
Matrix-Product: Threshold

Similar analysis $\rightarrow$ when to switch to the random strategy

![Graph showing the normalized communication amount vs. value of $\beta$.](image)
Matrix Product

Matrix-Product: Simulations

![Graph showing normalized communication amount vs number of processors]

- $N = 100$ blocks, $1,000,000$ tasks.
Outline

Introduction

Outer Product

Matrix Product

Extensions

Conclusion and Perspectives
Possible Extensions

- Joint works with E. Agullo, L. Eyraud-Dubois, A. Guermouche, J. Hermann, S. Kumar, T. Lambert, L. Marchal, S. Thibault
- Outer product and matrix multiplication
- Current limitations
  1. Why not using some static knowledge
  2. Still independent tasks sharing input files

1. Static vs Dynamic
   - Purely dynamic strategy (agnostic to processor speeds)
   - Can we inject static knowledge?

2. Data dependencies
   - From independent tasks sharing RO input files
   - To tasks sharing RW input/output files
Are static strategies so bad (I)?

- Static strategies are bad...
  - Need to be able to cope with failures
  - Need to be able to cope with unexpectedly slow resources
- But they are clever!
  - The ratio is close to 1 is nothing unexpected happens
- Idea: mix both strategies
  - Very preliminary tests (Thomas Lambert)
  - H1: static solution + work stealing at the end
  - H2: static solution + work stealing + last tasks replicated
- Rationale
  - H2 induces more communications
  - But may lead to large termination time (unrealistic)
Are static strategies so bad (II)?

- H1: static solution + work stealing at the end
- H2: static solution + work stealing + last tasks replicated

- 20 processors (min 1.1)
  - H2 better than H1
  - H2 >>> Dynamic if relative error on speeds < 15%

- 50 processors (min 1.05)
  - H2 better than H1
  - H2 >>> Dynamic if relative error on speeds < 10%
Extensions to More Complex Kernels

- Cholesky decomposition on CPU/GPU platforms
- Rules of thumb:
  - First and last POTRFs on GPUs
  - No GEMMs on CPUs, as many TRSMs as possible
  - Allocate remaining tasks using dynamic strategies

- Static
  - Solve a linear program that says what processors should spend their time to do without data dependences

- Dynamic
  - Describe the evolution of the processed boundary as a dynamic system
Conclusion and Perspectives

Outline

Introduction

Outer Product

Matrix Product

Extensions

Conclusion and Perspectives
Conclusion

What we have done:

- Design and analyze simple randomized dynamic strategy
- Analysis enables to tune strategy’s parameters (threshold)
- For outer and matrix-product (data dependencies only)
What remains to be done!

- A more formal proof
  - Mean Field Approximation?
- Better mix static and dynamic strategy
  - Estimation of processor speeds
  - Use a (quasi-optimal) static strategy for $X\%$ of the tasks
  - Allocate remaining tasks using dynamic strategies
  - Model enables to determine $X$
- Move to more complex kernels with task dependencies
  - Cholesky decomposition on CPU/GPU platforms
- Ultimate goal
  - Given a platform and a task graph
  - Design a set of scheduling policies and incentives
  - Such that dynamic scheduler perform as well as best offline schedulers (post mortem execution).
Thank You

Olivier.Beaumont@inria.fr, Loris.Marchal@ens-lyon.fr