A runtime approach to dynamic resource allocation for sparse direct solvers

Andra Hugo, Abdou Guermouche Pierre-André Wacrenier, Raymond Namyst
Inria, LaBRI, University of Bordeaux
The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which requires a big programming effort, is difficult to maintain and update, and is prone to (performance) portability issues.
General context

- Runtimes systems provide an abstraction layer that hides the architecture details
- The workload is expressed as a DAG of tasks where the dependencies are:
  - defined explicitly
  - defined through rules
  - automatically inferred
- The scheduler decides when/where to execute a task
- The drivers deploy the code on the devices
- The memory manager does the memory transfers and guarantees the consistency
Motivation application

The qr_mumps sparse solver

The multifrontal QR factorization is guided by a graph called elimination tree

- Five elementary operations:
  - Activate
  - Panel
  - Update
  - Assemble
  - Clean
Motivation application

The qr_mumps sparse solver

The multifrontal QR factorization is guided by a graph called elimination tree

• Data-flow parallel approach
  - Tasks are operations on portion of fronts (1-D partitioning)
  - Tasks are scheduled dynamically (dependencies between them)

• Tree of DAGs:
  - Nodes = DAG
  - Edges = dependencies
DAG structure of the parallel applications:

- Use data-flow approach:
  - DAG of sequential tasks
- Advantages:
  - Fine granularity
  - Increased parallelism
- Drawbacks for big DAGs:
  - Overhead of the runtime
  - Complexity of the scheduling

Submitted to the runtime

Runtime

Drivers (CUDA, OpenCL)

- CPUs
- GPUs
- MICs
DAG structure of the parallel applications:

- Use data-flow approach:
  - DAG of sequential tasks

- Advantages:
  - Fine granularity
  - Increased parallelism

- Drawbacks for big DAGs:
  - Overhead of the runtime
  - Complexity of the scheduling

- Possible solution:
  - Pack sub-DAGs into bigger tasks: malleable tasks
  - Use high-level scheduling algorithm

Submitted to the runtime
Interaction between the application & the runtime

The runtime doesn’t “automatically” ...

• Improve the performance of the application:
  - Structure the DAG (application)
  - Map it to the topology (runtime)
    ⇒ Enhance locality
    ⇒ Respect the critical path
• Difficulties:
  - What branches of the DAG to merge
  - Allocate resources for them
Using StarPU as an experimental platform to study resource negotiation

• The StarPU runtime system
  - Dynamically schedule tasks on all processing units
  • See a pool of heterogeneous processing units

- Avoid unnecessary data transfers between accelerators
  • Software VSM for heterogeneous machines

- Open scheduling platform
  • Different schedulers to meet different needs
Scheduling Contexts: to manage parallel tasks

- Isolate concurrent parallel codes
  - “lightweight virtual machines”
Scheduling Contexts: to manage parallel tasks

- Isolate concurrent parallel codes
  - “lightweight virtual machines”

- Contexts may expand and shrink
  - Hypervised approach
    - Resize contexts
    - Share resources
  - Maximize overall throughput
  - Use dynamic feedback both from application and runtime
    - Monitor the PUs
    - Monitor the application
Hierarchical parallelism in qr_mumps

- Idea:
  - Split the set of PUs among the branches
  - Consider their workload
  - Assign all PUs to at least one subtree

Proportional mapping
Hierarchical parallelism in qr_mumps

- Idea:
  - Split the set of PUs among the branches
  - Consider their workload
  - Assign all PUs to at least one subtree

- Extension of the algorithm:
  - Stopping criterion for the top-down process
  - Bundles: set of PUs sharing a level of memory
Hierarchical parallelism in qr_mumps

- Balance the workload of the tasks
- DAG of malleable tasks
- Provide to the runtime:
  - The hierarchy of parallel tasks
  - The workload of the parallel tasks
    - Estimates built during analysis
    - Dynamic updates during factorization
Resize hierarchically the scheduling contexts

The role of the Hypervisor

- Monitors the PUs
- Information coming from the leaves towards the root:
  - Efficiency of the PUs
  - Speed of the scheduling contexts

- Resizes the contexts locally
  - Resizing decisions at each level
  - Deadlines per children sharing the same parent
Allocate processing units to the contexts

By predicting the future

• **Input**: the workload (number of flops) of each context
• Rough computation of the number of resources needed by each context
  - How many CPUs allocated to each context?

\[
\max \left( \frac{1}{t_{max}} \right) \text{ subject to } \left( \forall c \in C, n_{\alpha,c}v_{\alpha,c} \geq \frac{W_c}{t_{max}} \right) \wedge \left( \sum_{c \in C} n_{\alpha,c} = n_{\alpha} \right) \wedge \left( \forall c \in C, n_{\alpha,c} < \max_{\alpha,c} \right)
\]
Allocate processing units to the contexts
By predicting the future

- **Input:** the workload (number of flops) of each context
- Rough computation of the number of resources needed by each context
  - How many CPUs allocated to each context?

\[
\max \left( \frac{1}{t_{\max}} \right) \quad \text{subject to} \quad \begin{align*}
\forall c \in C, n_{\alpha,c} & \leq \frac{W_c}{t_{\max}} \\
\sum_{c \in C} n_{\alpha,c} & = n_{\alpha} \\
\forall c \in C, n_{\alpha,c} & < \max_{\alpha,c}
\end{align*}
\]
Allocate processing units the tree

By predicting the future

- **Input**: the workload (number of flops) of each context
- Rough computation of the number of resources needed by each context
  - How many CPUs allocated to each context?

\[
\text{max} \left( \frac{1}{t_{\text{max}}} \right) \text{ subject to } \\
\left( \forall C, \sum n_{\alpha,c} \leq n_{\alpha} \right) \\
\left( \forall C, n_{\alpha,c} < \max_{\alpha,c} \right) \\
\left( \forall C, n_{\alpha,c} = \frac{W_c}{t_{\text{max}}} \right)
\]
Allocate processing units to the contexts

By predicting the future

• **Input**: the workload (number of flops) of each context
• Rough computation of the number of resources needed by each context
  - How many CPUs allocated to each context?

\[
\text{max} \left( \frac{1}{t_{\text{max}}} \right) \quad \text{subject to}
\]

\[\forall c \in C, n_{\alpha,c}, v_{\alpha,c} \geq \frac{W_c}{t_{\text{max}}} \]

\[\sum_{c \in C} n_{\alpha,c} = n_{\alpha} \]

\[\forall c \in C, n_{\alpha,c} < \max_{\alpha,c} \]

- Execution time of the application
- Speed of CPUs in Context C
- Rough information about the parallelisme of the kernels

\[\text{Workload of Context } c\]
\[\text{nCPUs in Context } c\]
Triggering the reallocation of resources
When static dimensioning is not enough

• The hypervisor monitors:
  - Idle PUs
  - Execution speed of the contexts
    • Execution time of the application cut in intervals
    • Observed Speed = executed_flops / time
    • Target speed = #PUs * average speed of PU

• The hypervisor:
  - Iterates hierarchically the tree of contexts (root -> leaves)
  - Searches for idle PUs or slow contexts
  - Stops at the level where the application doesn’t behave “well”
  - Triggers resizing hierarchically starting from that level
Using contexts to guide scheduling
Dynamically assigning PUs to the parallel nodes of the tree
Scalability of the parallel tasks

- Idleness problem:
  - Sequential elementary tasks
  - Correct nCPUs but not enough tasks
- Possible solution:
  - Use the idle time to compute a max
  - Drawback:
    - When to increase it?
    - What to do with the unused PUs?
- Intuition:
  - Use parallel/moldable elementary tasks to approximate malleable tasks
  - Need to find a good tradeoff between inner and outer parallelism
Using contexts to guide scheduling

Efficiency gain: on small problems
Using contexts to guide scheduling

Efficiency gain: on large problems

![Graph showing time with respect to the version not using contexts for different numbers of cores used.]
Using contexts to guide scheduling

Improve locality for Rucci1
Conclusion

• Structure the parallelism of the application
  - By building a hierarchy of the scheduling contexts

• Use the hypervisor in order to:
  - Monitor the efficiency of the PUs
  - Monitor the speed of the scheduling contexts
  - Dynamically resize the scheduling contexts

• Improve the behavior of qr_mumps:
  - By enforcing the locality
  - By respecting the critical path

• Need a strong interaction between the solver and the hypervisor
On going work (1/2)

- Deal with non-StarPU tasks
  - Sub-DAGs of StarPU tasks
  - Parallel tasks (parallel mkl blas, …)

- Resizing StarPU/OpenMP/TBB contexts
  - Common metrics?

- Contexts as a way to better utilize Heterogeneous/Manycore Architectures
  - GPUs
  - Intel Xeon Phi accelerators
On going work (2/2)

• Increase the amount of parallelism
  - Move to 2D partitionning of frontal matrices when needed.

• Limit the memory usage of the factorization
  - Control task submission while avoiding deadlocks.

• Consider different paradigms (e.g. PTG model)
  - A ParSEC-based version of the solver is being developed.

• Exploit accelerator-based heterogeneous architectures
  - GPU, Intel Xeon-Phi, …
  - Still preliminary.
  - Need for scheduling algorithms for graphs of malleable/moldable tasks running on heterogeneous platforms.

• Study distributed memory architectures.