Introduction to Principles of Task-Based Performance Portability
Illustration with the StarPU Runtime System for Heterogeneous HPC Platforms

Olivier Aumage

Team STORM
Inria — LaBRI
Bordeaux, France

ETP4HPC Webinar – 2022 / 03 / 18
Task-Based Performance Portability in HPC

Maximising long-term investments in a fast evolving, complex and heterogeneous HPC landscape

- Authors
  - Olivier Aumage, Inria
  - Paul Carpenter, Barcelona Supercomputing Center
  - Siegfried Benkner, the University of Vienna

- 2021/10/05 – etp4hpc.eu
  - DOI: 10.5281/zenodo.5549731
Task-Based Performance Portability in HPC

Key insights

- Porting applications is difficult and must be successful in a short time
Task-Based Performance Portability in HPC

Key insights

- Porting applications is difficult and must be successful in a short time
Task-Based Performance Portability in HPC

Key insights

- Porting applications is difficult and must be successful in a short time.
Task-Based Performance Portability in HPC

Key insights

- Porting applications is difficult and must be successful in a short time

- Applications should therefore be expressed in a way that facilitates performance portability
Task-Based Performance Portability in HPC

Key insights

- Porting applications is difficult and must be successful in a short time

- Applications should therefore be expressed in a way that facilitates performance portability

- Task-based programming models allow HPC programmers to express applications and workflows in such a performance portable way
High-Performance Computing

Supercomputers Hardware Evolution

- **Fast paced**
  - Short lifetime: 5 – 10 years
- **Increasing complexity**
  - RIKEN Fugaku Computer: ~160K nodes, ~7M cores
- **Increasing heterogeneity**
  - Accelerators devices, FPGA, processing offload
- **Increasingly diverse purposes and designs**
  - Graph / Green / Top 500, HPCG

<table>
<thead>
<tr>
<th>Name</th>
<th>Start year</th>
<th>Performance (PFLOPS)</th>
<th>TOP500 ranking</th>
<th>CPU/GPU vendor</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fugaku</td>
<td>2020</td>
<td>415</td>
<td>June 2020 1st</td>
<td>Fujitsu</td>
<td>A64FX</td>
</tr>
<tr>
<td>Summit</td>
<td>2018</td>
<td>148</td>
<td>June 2018 to November 2019 1st</td>
<td>IBM, NVIDIA</td>
<td>POWER9, Tesla</td>
</tr>
<tr>
<td>Sierra</td>
<td>2018</td>
<td>94</td>
<td>November 2018 to November 2019 2nd</td>
<td>NRCPC</td>
<td>Sunway SW62010</td>
</tr>
<tr>
<td>Sunway TaihuLight</td>
<td>2016</td>
<td>93</td>
<td>June 2016 to November 2017 1st</td>
<td>Fujitsu</td>
<td>SPARC64 VIIIfx</td>
</tr>
<tr>
<td>K</td>
<td>2011</td>
<td>10</td>
<td>June 2011 – November 2011 1st</td>
<td>Fujitsu</td>
<td>SPARC64 VIIIfx</td>
</tr>
</tbody>
</table>

[Wikipedia.org: Fugaku vs some former rank #1 Top500 supercomputers](https://en.wikipedia.org/wiki/Fugaku)
Proposal

Programming for performance portability

- Focus on expressing work instead of managing workers
  - Rely on abstractions instead of hardware dependent work divisions
Proposal

Programming for performance portability

- Focus on expressing work instead of managing workers
  > Rely on abstractions instead of hardware dependent work divisions
Proposal

Programming for performance portability

- Focus on expressing work instead of managing workers
  - Rely on abstractions instead of hardware dependent work divisions
Programming for performance portability

- **Focus on expressing work instead of managing workers**
  - Rely on abstractions instead of hardware dependent work divisions

- **Confine adaptation effort to select kernel routines**
  - Tasks
Proposal

Programming for performance portability

- Focus on expressing work instead of managing workers
  > Rely on abstractions instead of hardware dependent work divisions

- Confine adaptation effort to select kernel routines
  > Tasks
Proposal

Programming for performance portability

- **Focus on expressing work instead of managing workers**
  - Rely on abstractions instead of hardware dependent work divisions

- **Confine adaptation effort to select kernel routines**
  - **Tasks**

![Diagram showing task-based performance portability with StarPU](image)
Programming for performance portability

- Focus on expressing work instead of managing workers
  > Rely on abstractions instead of hardware dependent work divisions

- Confine adaptation effort to select kernel routines
  > Tasks
Proposal

Programming for performance portability

- **Focus on expressing work instead of managing workers**
  - Rely on abstractions instead of hardware dependent work divisions

- **Confine adaptation effort to select kernel routines**
  - Tasks

- **Easier / safer / more efficient application development**
  - for human programmers

- **Easier / safer / more effective application analysis and optimization**
  - for tools
Task-based Parallel Programming

Principles

- **Separate multiple concerns**
  - General application algorithmics
  - Low-level task kernel optimization
  - Resource management and work assignment

- **Maximise long term investments**
  - Mostly fixed application structure
    - Long term stability
  - [Machine- | Device-] specific routines == **Tasks**
    - Short term, localized optimization effort
Task-based Parallel Programming

**Principles**

- **Separate multiple concerns**
  - General application algorithmics
  - Low-level task kernel optimization
  - Resource management and work assignment

- **Maximise long term investments**
  - Mostly fixed application structure
    - Long term stability
  - \([ \text{Machine-} \mid \text{Device-} ]\) specific routines = Tasks
    - Short term, localized optimization effort

- **Model maturity**
  - Cilk, *Blumofe et al, 1995*
  - OpenMP 3.0 standard, 2008

- **Active research ecosystem**
Task-based Parallel Programming

Principles

- **Separate multiple concerns**
  - General application algorithmics
  - Low-level task kernel optimization
  - Resource management and work assignment

- **Maximise long term investments**
  - Mostly fixed application structure
    - Long term stability
  - [Machine- | Device-] specific routines == Tasks
    - Short term, localized optimization effort

- **Model maturity**
  - Cilk, Blumofe et al, 1995
  - OpenMP 3.0 standard, 2008

- **Active research ecosystem**

---

- **StarPU**
  - Inria / LaBRI, Bordeaux, 2009

- **DuctTeip / SuperGlue**
  - University of Uppsala, 2013

- **HPX**
  - Louisiana State University, 2013

- **OCR**
  - Specification, 2014
  - Several implementations
    - Intel+Rice University
    - University of Vienna

- **OmpSs**
  - BSC, 2008

- **PaRSEC**
  - ICL / UTK, 2012

- **Regent / Legion**
  - Stanford, 2012

- ...and many others...
Task-based Parallel Programming

Principles

- **Separate multiple concerns**
  - General application algorithmics
  - Low-level task kernel optimization
  - Resource management and work assignment

- **Maximise long term investments**
  - Mostly fixed application structure
    - Long term stability
  - [ Machine- | Device- ] specific routines == Tasks
    - Short term, localized optimization effort

- **Model maturity**
  - Cilk, Blumofe et al, 1995
  - OpenMP 3.0 standard, 2008

- **Active research ecosystem**

---

- **StarPU**
  - Inria / LaBRI, Bordeaux, 2009

- **DuctTeip / SuperGlue**
  - University of Uppsala, 2013

- **HPX**
  - Louisiana State University, 2013

- **OCR**
  - Specification, 2014
  - Several implementations
    - Intel+Rice University
    - University of Vienna

- **OmpSs**
  - BSC, 2008

- **PaRSEC**
  - ICL / UTK, 2012

- **Regent / Legion**
  - Stanford, 2012

- ... and many others ...
The StarPU runtime system

Task-based Computing Runtime System

- Inria / LaBRI, Bordeaux, 2009
  - PhD Cédric Augonnet

- Task scheduling on a heterogeneous, accelerated node
  - General purpose CPU cores
  - Specialized accelerators
    - Discrete board + embedded memory

Heterogeneous computing node
The StarPU runtime system

Task-based Computing Runtime System

- Inria / LaBRI, Bordeaux, 2009
  - PhD Cédric Augonnet

- Task scheduling on a heterogeneous, accelerated node
  - General purpose CPU cores
  - Specialized accelerators
    - Discrete board + embedded memory

- Usage
  - Direct programming from application
    - C, C++, Fortran
  - Compiler / Language
    - OpenMP, Julia, Python
  - Parallel numerical libraries
    - Inria Solverstack

---

HPC Applications

- CHAMELEON
  - Dense linear algebra
- PaStiX
  - Sparse linear algebra
- ScALFMM
  - Fast multipole method

---

StarPU Runtime System

Drivers (CPU, CUDA, OpenCL, ...)

- CPU
- GPU
- ...
Programming model

Tasks + data dependencies

- **Tasks**
  - Annotated kernels
  - → Potential parallelism

- **Data dependencies**
  - Set of constraints
    - Input needed
    - Output produced
  - → Degrees of freedom

\[
\begin{align*}
\text{Task} &= \text{kernel} + \text{data dependencies} \\
A &= A + B \\
\end{align*}
\]
Programming model

Sequential Task Flow

- Tasks submitted sequentially
  - Deferred execution

- Dependence graph built incrementally
  - Vertex == task
  - Edge == data dependence

```c
for (j = 0; j < N; j++)
{
    starpu_task_insert( POTRF (RW, A[j][j]) );
    for (i = j+1; i < N; i++)
        starpu_task_insert( TRSM (RW, A[i][j], R, A[j][j]) );
    for (i = j+1; i < N; i++)
        starpu_task_insert( SYRK (RW, A[i][i], R, A[i][j]) );
    for (k = j+1; k < i; k++)
        starpu_task_insert( GEMM (RW, A[i][k], R, A[i][j], R, A[k][j]) );
}
starpu_task_wait_for_all();
```

Flow of task submissions
Making hardware dependent decisions on behalf of the programmer

StarPU execution model

- **Scheduling engine**
  - Programmable policies
    - Theoretical algorithmic corpus
  - Task mapping
    - Anticipative (== planning)
    - Reactive (== work stealing)

- **Distributed Shared Memory (DSM) engine**
  - Data management
  - Data replication and consistency

- **Performance modeling engine**
  - Task execution time inference
  - Data transfer time inference

Mapping a task graph on hardware resources
Heterogeneous processing resource management

Dynamically planned execution

- **Kernel performance estimation**
  - Per device
  - Per routine variant
  - Per input size

- **Task execution time inference**
  - History-based
  - Custom cost function

- **Data transfer time inference**
  - Bus sampling

---

C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier
*StarPU: a unified platform for task scheduling on heterogeneous multicore architectures*
Hierarchical tasks — Bubbles

- **Contribution**
  > PhD Gwenolé Lucas

- **Notion of "task bubble"**
  > **Task view**: macro task
  > **Bubble view**: DAG of micro tasks

- **Adaptiveness**
  > Granularity / Level of parallelism
  > Versioning
  > JIT view selection

- **Scalability**
  > Controlled DAG discovery
Resource management for multiple task graphs

Scheduling contexts

- **Contribution**
  - PhD Andra Hugo

- **Single StarPU instance**
  - Multiple task graphs
  - Concurrent StarPU-based routines
  - Composition

- **Dynamic resource assignment**
  - Malleability
    - CPU cores
    - Accelerator devices

A. Hugo, A. Guermouche, P.-A. Wacrenier, R. Namyst
*Composing multiple StarPU applications over heterogeneous machines: A supervised approach*
Resource management among multiple task-based runtimes

**Nested composition**

- Task parallel application or library + parallel kernel tasks
  - Offload and resource enforcement API

- Project H2020 INTERTWinE — http://www.intertwine-project.eu/
Resource management among multiple task-based runtimes

Concurrent composition

- One parallel application or library, concurrent to another parallel library
  - Dynamic Resource Sharing (DRS) API

- Project H2020 INTERTWinE — [http://www.intertwine-project.eu/](http://www.intertwine-project.eu/)
Resource management among multiple task-based runtimes

Concurrent composition

- **Parallel application or library // parallel library**
  - Dynamic Resource Sharing (DRS) API

- **Direct interfacing**
  - StarPU
  - OmpSs
  - Same process computing resource sharing

- **Interfacing through external component**
  - DLB (Dynamic Load Balancing) framework
    - Developed at BSC
    - Library + external daemon
  - Same process or multi-processes computing resource sharing

- **Project H2020 INTERTWinE** — [http://www.intertwine-project.eu/](http://www.intertwine-project.eu/)
Distributed processing management

**StarPU-MPI**

- **Contributions**
  - Early prototype by Cédric Augonnet
  - PhD Marc Sergent

- **Two execution models supported**
  - Master – workers
  - Fully distributed

Cluster of heterogeneous nodes
Fully-distributed model

No master node

- Local task graph discovery
  - Whole graph discovered on every node
  - Initial data distribution given by application

- Local decisions
  - Task execution
    - Data ownership
  - Data transfers
    - Internode edges

Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model
Failure Tolerance

Task graph-based checkpointing

- **Contribution**
  - PhD Romain Lion

- **Distributed data replication**
  - Leverage task graph knowledge
  - Replicate on neighbor nodes
  - Adapt number of replicates to desired robustness

- **Restart**
  - Restart tasks from data replicates on surviving nodes

StarPU Wrap-up

The StarPU task-based runtime system

- Comprehensive in-app resource management
  - Heterogeneous processing units: CPU, GPU, ..., *PU, FPGA
    - Planned + work stealing task scheduling
    - Performance modeling
  - Heterogeneous memory resource management
    - Data replication + memory consistency
    - NUMA, HBM, on-device memory, out-of-core
  - Ecosystem friendly resource management
    - Interoperability, composability, malleability

Take-away

Applications need to be expressed in a way that facilitates high performance across a range of hardware and situations

- **Programmer-friendly, tools-friendly applications**
  - Platform-independent, malleable, asynchronous

- **Stable interfaces**
  - Between applications / libraries / kernels / platform-tuned software

- **Task abstraction**
  - Basis for composable and dynamic performance portable interfaces

- **Greater degree of trust from applications**
  - Structured parallel programming
Thanks!