Analyzing Dynamic Task-Based Applications on Hybrid Platforms: An Agile Scripting Approach

Vinícius Garcia Pinto, Luka Stanisic, Arnaud Legrand, Lucas Schnorr, Samuel Thibault, Vincent Danjean

Visual Performance Analysis workshop
SuperComputing 2016, Salt Lake City
November 18, 2016
Context

Current HPC architectures

- Moving from transistors to heterogeneity scaling
- Plethora of hybrid computing resources
  (CPUs, GPUs, MICs)
Context

Current HPC architectures
- Moving from transistors to heterogeneity scaling
- Plethora of hybrid computing resources (CPUs, GPUs, MICs)

Programming hybrid platforms
- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, ...)
  - Perfect control $\leadsto$ maximal achievable performance
Context

Current HPC architectures

- Moving from transistors to heterogeneity scaling
- Plethora of hybrid computing resources (CPUs, GPUs, MICs)

Programming hybrid platforms

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, ...)
  - Perfect control $\rightarrow$ maximal achievable performance
  - Monolithic codes $\rightarrow$ hard to develop and maintain
  - Hard to optimize $\rightarrow$ performance portability
  - Fixed scheduling $\rightarrow$ sensitive to variability
Context

Current HPC architectures

- Moving from transistors to heterogeneity scaling
- Plethora of hybrid computing resources (CPUs, GPUs, MICs)

Programming hybrid platforms

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, ...)
  - Perfect control $\implies$ maximal achievable performance
  - Monolithic codes $\implies$ hard to develop and maintain
  - Hard to optimize $\implies$ performance portability
  - Fixed scheduling $\implies$ sensitive to variability

- Recent task-based programming models (PaRSEC, OmpSs, Charm++, StarPU, ...)
  - Single, abstract programming model based on DAG
  - Runtime responsible for dynamic scheduling
  - Portability of code and performance
Context

Current HPC architectures

- Moving from transistors to heterogeneity scaling
- Plethora of hybrid computing resources (CPUs, GPUs, MICs)

Programming hybrid platforms

- Traditional, explicit programming models (MPI, CUDA, OpenMP, pthreads, ...)
  - Perfect control $\leadsto$ maximal achievable performance
  - Monolithic codes $\leadsto$ hard to develop and maintain
  - Hard to optimize $\leadsto$ performance portability
  - Fixed scheduling $\leadsto$ sensitive to variability

- Recent task-based programming models (PaRSEC, OmpSs, Charm++, StarPU, ...)
  - Single, abstract programming model based on DAG
  - Runtime responsible for dynamic scheduling
  - Portability of code and performance
  - New challenge $\leadsto$ scheduling heuristic
Performance evaluation challenges

- No clear phases
- Hidden idle time, spread everywhere
- Non-deterministic executions

How to improve performance if we do not even understand what happened?
Analyzing Scheduler Performance

Scheduler’s role

- Assigning tasks to resources (CPU vs GPU)
- Anticipating critical path
- Data movements

```c
for (k = 0; k < N; k++) {
    DPOTRF(RW, A[k][k]);
    for (i = k+1; i < N; i++)
        DTRSM(RW, A[i][k], R, A[k][k]);
    for (i = k+1; i < N; i++)
        DSYRK(RW, A[i][i], R, A[i][k]);
        for (j = k+1; j < i; j++)
            DGEMM(RW, A[i][j], R, A[i][k], R, A[j][k]);
}
```
Analyzing Scheduler Performance

Scheduler’s role
- Assigning tasks to resources (CPU vs GPU)
- Anticipating critical path
- Data movements

Key questions when analyzing traces
- Evaluate scheduling decisions
- Micro vs. macro analysis
- Exploit DAG (code) structure
- Compare to lower bounds
- How to compare two/several executions?

for (k = 0; k < N; k++) {
    DPOTRF(RW, A[k][k]);
    for (i = k+1; i < N; i++)
        DTRSM(RW, A[i][k], R, A[k][k]);
    for (i = k+1; i < N; i++)
        DSYRK(RW, A[i][i], R, A[i][k]);
    for (j = k+1; j < i; j++)
        DGEMM(RW, A[i][j], R, A[i][k], R, A[j][k]);
}
Visual Performance Analysis Tools

Current tools

- Implemented in C/C++ to scale
- Interactive and user friendly
- Large and complex source code, difficult to extend
- Generally not designed for hybrid platforms and dynamic runtimes
- Flexible filter calls for scripting capability
- Lack custom views exploiting application and platform structure
Visual Performance Analysis Tools

Current tools

- Implemented in C/C++ to scale
- Interactive and user friendly
- Large and complex source code, difficult to extend
- Generally not designed for hybrid platforms and dynamic runtimes
- Flexible filter calls for scripting capability
- Lack custom views exploiting application and platform structure

Our proposal

- Builds on modern, generic data analysis tools
- Modular scripting approach: `paje_dump + R + ggplot + plotly (~ 1K SLOC)`
- Allows to easily create rich visualizations and jump between views
Tiled Cholesky factorization

for (k = 0; k < N; k++) {
    DPOTRF(RW, A[k][k]);
    for (i = k+1; i < N; i++)
        DTRSM(RW, A[i][k], R, A[k][k]);
    for (i = k+1; i < N; i++)
        DSYRK(RW, A[i][i], R, A[i][k]);
    for (j = k+1; j < i; j++)
        DGEMM(RW, A[i][j], R, A[i][k], R, A[j][k]);
}
Customized Visualizations

Large matrix
Customized Visualizations

Large matrix

Medium matrix + interaction
Conclusion and Future Work

Achievements

- Lightweight scripting approach for analyzing dynamic task-based applications on hybrid platforms
- Suitable for scheduling specialists
- General methodology combined with application/machine specific, adaptable extensions

Ongoing work

- Exploit complex application structures (qr_mumps)
- Spatial/temporal aggregation (Ocelotl)
- Investigate large scale distributed executions (StarPU-MPI)
Thank you!

Open science

- Git + org-mode methodology for better reproducibility

http://perf-ev-runtime.gforge.inria.fr/vpa2016/