Leveraging PEPPHER Technology for Performance Portable Supercomputing

Christoph Kessler*, Usman Dastgeer*, Mudassar Majeed*, Nathalie Furmento†, Samuel Thibault†, Raymond Namyst†, Siegfried Benkner‡, Sabri Pllana‡, Jesper Larsson Träff§, Martin Wimmer§
*Linköping University, S-58183 Linköping, Sweden. Email: firstname.lastname@liu.se
†University of Bordeaux, INRIA, Bordeaux, France. Email: firstname.lastname@labri.fr
‡University of Vienna, Austria. Email: firstname.lastname@univie.ac.at
§Vienna University of Technology, Austria. Email: {traff,wimmer}@par.tuwien.ac.at

***Abstract*—PEPPHER is a 3-year EU FP7 project that develops a novel approach and framework to enhance performance portability and programmability of heterogeneous multi-core systems. Its primary target is single-node heterogeneous systems, where several CPU cores are supported by accelerators such as GPUs. This poster briefly surveys the PEPPHER framework for single-node systems, and elaborates on the prospectives for leveraging the PEPPHER approach to generate performance-portable code for heterogeneous multi-node systems.**

I. The PEPPHER Framework for Heterogeneous Single-Node Systems

The PEPPHER project[1] is concerned with the problem of performance portability for heterogeneous many-core systems, and develops a novel framework for incremental development of performance portable code. This framework [2] consists of a component model, a composition and transformation layer, libraries of auto-tunable algorithms and data structures, and a run-time system. PEPPHER provides for (enhanced) performance portability by enabling static and dynamic selection of the most appropriate (highly performing, efficient) variant for any given properly componentized part of the application.

A PEPPHERed application consists of a main program plus a collection of annotated user-level *components*, where a component contains an interface of a (C/C++) function and one or several implementation variants of that function that may differ in their execution target (e.g. CPU or GPU), their programming model (e.g. OpenMP, CUDA, OpenCL, sequential C, or other C-based programming models), algorithms used, etc. Interfaces and implementation variants are annotated by XML descriptors that contain metadata about data types and access modes, deployment commands, explicit dependences on hardware and software, resource requirements, performance models, tunable parameters etc. The metadata make PEPPHER components resource- and performance-aware, facilitating component variant selection and scheduling for heterogeneous architectures. For passing parameters to or from components one can use native C/C++ types or special, generic *smart container types* (e.g., Vector and Matrix) that provide additional optimizations for avoiding unnecessary memory transfers to/from GPU device memory or for data-flow driven asynchronous execution.

Component invocations result in tasks to be executed by the PEPPHER run-time system. A composition tool generates wrapper code from each component's descriptor metadata. The wrapper intercepts each call to the component and either directly delegates the selection of the expected fastest implementation variant(s) for the current call context to the run-time system, or pre-selects one or a few expected fastest implementation variant(s) for the current call context based on off-line performance modeling and tuning. In both cases the generated wrapper creates a PEPPHER task for the call to be scheduled by the PEPPHER run-time system. The run-time system will for each task perform final selection and scheduling of the implementation variant estimated to be most appropriate for the current execution context, based on run-time performance modeling.

The template library *SkePU* [4] provides pre-defined generic components, so-called *skeletons*, that implement frequently needed computation patterns and for which the component metadata is implicitly defined. SkePU provides the most common data-parallel skeletons (map, reduce, scan, mapoverlap, etc.) and a task-parallel skeleton (farm). Each skeleton comes with implementation variants for specific execution units (GPU vs. CPU, multi-GPU, multi-CPU) and specific programming models (CUDA, OpenCL, OpenMP, and sequential C), but also with a generic back-end that delegates the implementation variant selection to the PEPPHER run-time system [3]; hence, SkePU can be used both stand-alone and within the PEPPHER framework. Like user-defined PEPPHER components, SkePU skeletons are tunable, e.g., a skeleton can select the expected fastest variant depending on the current call context or to select expected best values for skeleton-specific or device-specific tunable parameters. SkePU allows for hybrid execution if smart containers are used [5].

The PEPPHER run-time system, which is based on *StarPU* [1], dynamically schedules tasks over the various heterogeneous processing units in the system. Tasks can have multiple implementations, including several variants for a given type of processing unit. Task input data are registered in the system which keeps track of copies in memory nodes to minimize data transfers and perform data prefetching. The load

[1]This research has been partly funded by EU FP7 project PEPPHER, www.peppher.eu, grant #248481, and by SeRC.

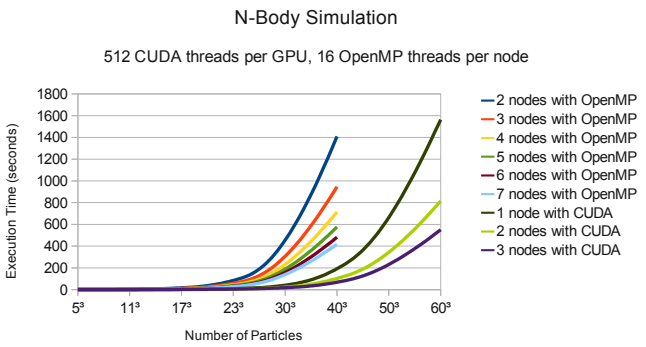

Fig. 1: Execution times of a simple SkePU application (100 time steps of a N-body simulation with force calculation expressed by the *maparray* skeleton), with MPI+OpenMP and MPI+CUDA backends using 1 to 8 resp. 1 to 3 nodes of a GPU cluster equipped with Xeon X5550 CPUs (8 cores per node, 2.67GHz) and Nvidia Tesla C2050 GPUs. Timings are averaged over the last 9 of 10 runs.

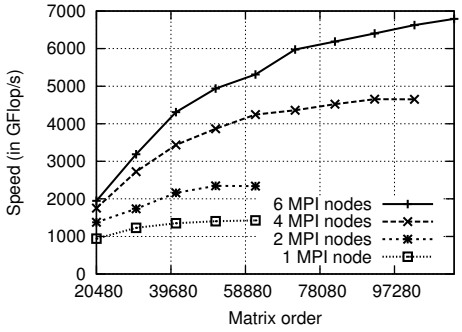

Fig. 2: Speedup of the "sequential-looking" Cholesky decomposition over a 6-node cluster of 6-core 2.67GHz Intel Nehalem X5650 machines equipped with 3 NVIDIA Fermi M2070 GPUs

balancing algorithm is fully configurable, and can build on auto-calibrated cost prediction mechanisms enabling accurate estimation of task duration.

A sequential legacy application can be componentized and annotated ("PEPPHERed") incrementally. (Almost) no changes of existing code are required, unless one wants to use smart containers or annotate individual calls. Performance portability is achieved because the tuning, selection and scheduling of implementation variants automatically adapts to new hardware configurations.

II. PEPPHER Technology in a Multi-Node Context

Although the PEPPHER framework has been developed for single-node, heterogeneous architectures, we believe that many of the ideas can be extended toward multi-node, heterogeneous systems. This will entail some change in the execution model and some restrictions in its implementation. For instance, communication between components must be delegated to communication interface like MPI, and component-variant selection must be done in a consistent way in order to avoid deadlocks. We here report on ongoing work in two directions.

A. Cluster-SkePU

The dataparallel skeletons with CUDA/OpenCL/OpenMP/C backends and the Vector container of SkePU have been extended to support clusters of heterogeneous nodes. A main challenge is that containers (Vector, Matrix...) must support several (HPF-like) data distributions and track where currently valid data partitions reside to perform the necessary communication automatically when required; communication is delegated to MPI.

Figure 1 shows an early result of Cluster-SkePU on a GPU cluster for an N-body simulation using the *maparray* skeleton for the force calculation between particles whose properties (positions, speeds etc.) are stored in SkePU Vector containers. The Cluster-SkePU implementation uses OpenMPI 1.5.3 and nvcc for the CUDA/MPI backend (using -lmpi); for the OpenMP/MPI backend the compiler used is icpc. For this simple application linear, relative speed-up in the number of nodes is achieved.

B. Cluster-StarPU

The StarPU run-time system [1] has been extended to integrate with MPI. The extensions follow two directions. In case an existing MPI application is modified to run an instance of StarPU on each node, helpers are provided to delegate transfers of data managed by StarPU to MPI. Since such data can be non-contiguous, and spread among different memory nodes, StarPU enforces the required memory synchronization and builds the appropriate MPI datatypes.

In the case the application obeys a task-based programming model, support is provided for extending existing single-node applications to obtain a cluster-enabled implementation. The application provides a data distribution over MPI nodes, and makes each node iterate over the set of tasks to be executed; StarPU then automatically infers, from task data requirements, which MPI communications are required. This allows a sequential-looking application code to run on heterogeneous clusters with CPUs and GPUs.

References

[1] C. Augonnet, S. Thibault, R. Namyst and P. A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009* **23**(2), Feb 2011.

[2] S. Benkner, S. Pllana, J. L. Träff, P. Tsigas, U. Dolinsky, C. Augonnet, B. Bachmayer, C. W. Kessler, D. Moloney and V. Osipov. PEPPHER: Efficient and Productive Usage of Hybrid Computing Systems. *IEEE Micro* **31**(5), September/October 2011.

[3] U. Dastgeer, C. Kessler and S. Thibault. Flexible runtime support for efficient skeleton programming. Proc. ParCo-2011 conference, Ghent, Belgium, Sep. 2011.

[4] J. Enmyren and C. Kessler: SkePU: A Multi-Backend Skeleton Programming Library for Multi-GPU Systems. Proc. 4th Int. Workshop on High-Level Parallel Programming and Applications (HLPP-2010), Baltimore, USA, Sep. 2010. ACM.

[5] C. W. Kessler, U. Dastgeer, S. Thibault, R. Namyst, A. Richards, U. Dolinsky, S. Benkner, J. L. Träff and S. Pllana: Programmability and Performance Portability Aspects of Heterogeneous Multi-/Manycore Systems. Proc. DATE-2012 conference on Design, Automation and Test in Europe, Dresden, March 2012. IEEE CS Press, pp. 1403-1408.