Towards Better Memory Management in Heterogeneous GEMM

Daniel Mishler, George Bosilca, Thomas Herault
Innovative Computing Laboratory, University of Tennessee

PaRSEC

A task-based runtime controller for heterogeneous architectures:
https://github.com/icldisco/parsec

DPLASMA

Dense linear algebra package for heterogeneous systems:
https://github.com/icldisco/dplasma

PaRSEC Animation Utils

A visualization package developed for use with PaRSEC
https://github.com/DSMishler/PaRSEC_Animate_Demo

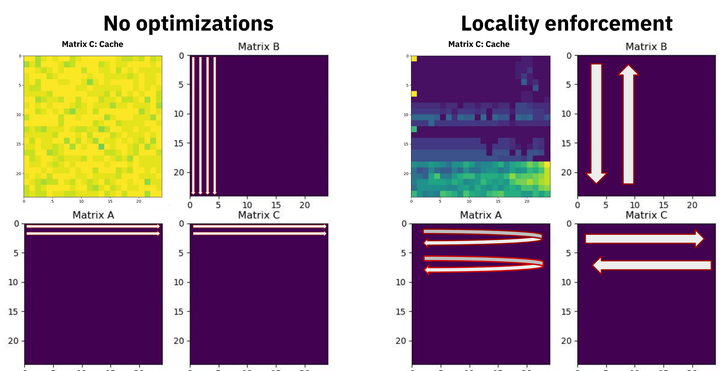
Schedulers are the future when it comes to heterogeneous architectures. As (even homogeneous) systems scale, it becomes more and more difficult to attain good cache reuse. PaRSEC is a task-based scheduler designed to be general, so it lends itself well to heterogeneous architectures. This poster focuses on one specific case of scheduling: using DPLASMA (A dense linear algebra package which calls PaRSEC) on General Matrix Multiplication in double precision (DGEMM).

With a scheduler, **time spent scheduling tasks is not spent performing work on said tasks**. What strategies can we consider to achieve the best decisions for schedulers with respect to memory locality while also minimizing the overhead of the scheduler?

GOAL

Improve & understand data reuse in a dynamic runtime; be **portable** & **scalable**.

Below is an example demonstrating PaRSEC getting data reuse. See how under Methodology. (yellow = memory never reused, blue = memory always reused)



PRIORITY CONTROLLED BINDING SCHEDULER

Every task in PaRSEC has a priority. Depending on the scheduler, this priority is strictly followed, taken as a hint, or outright ignored. The Priority Controlled Binding (PCB) scheduler masks out some of the priority bits and uses them to enforce which thread group that task is allowed to run on. Thread group 0 is all thread groups to reduce restrictiveness.

Methodology

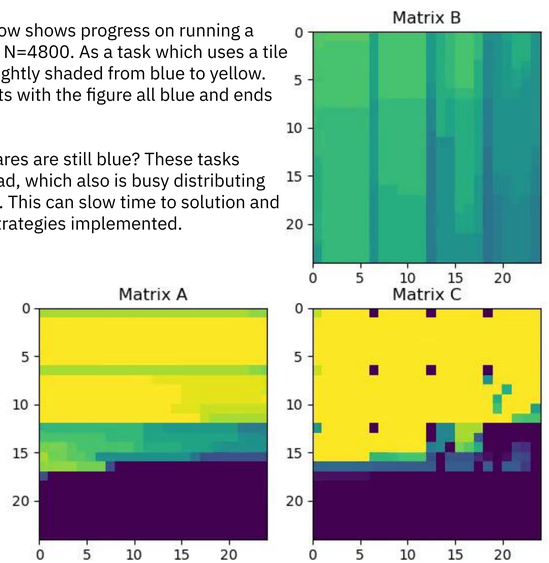
For multicore systems, ensure that tasks are handled by the runtime with appropriate constraints to ensure that a given core is likely to handle the same memory.

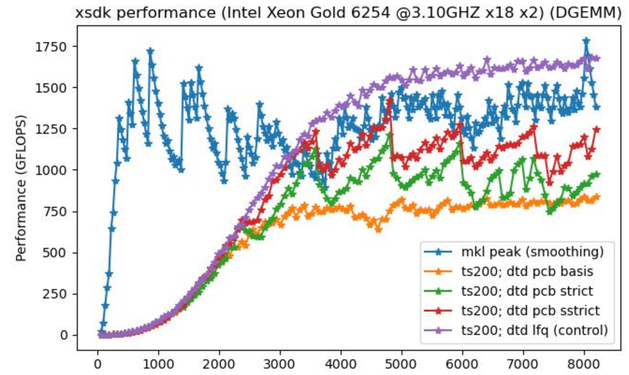
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12
13	14	15	16	13	14	15	16

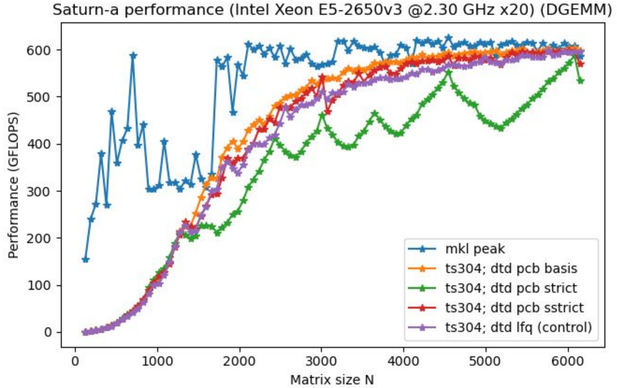
One application of the PCB scheduler is to paste a template across Matrix C that forces tasks which update a tile m, n to all run on the same core. This causes PaRSEC to enforce an order of execution like the one seen in the bottom left of this poster.

The figure to the right/below shows progress on running a matrix DGEMM of ts=200, N=4800. As a task which uses a tile is completed, the tile is slightly shaded from blue to yellow. The visualization tool starts with the figure all blue and ends with the figure all yellow.

Why do we see some squares are still blue? These tasks belong to the master thread, which also is busy distributing tasks to the other threads. This can slow time to solution and is a consideration in the strategies implemented.







- **LFQ**: Local Flat Queue, the default scheduler of PaRSEC
- **PCB**: Priority Controll Binding scheduler
- **PTG**: Parameterized Task Graph DAG builder
- **DTD**: Dynamic Tasks Discovery DAG builder

Performance

Our results show that when it comes to task-based scheduling, we can achieve low overhead while maintaining high cache reuse. We can have the best of both worlds with implementations that are light-handed with enforcement. For example, the **sstrict** (semistrict) implementation allows the other threads to assist the master thread as computation approaches a close.

Our work is not done yet, but we can summarize achievements and directions: we have achieved finer control over scheduling and our performance can compete with MKL on new Intel processors. While we perfect our basis implementations, we can look forward to heterogeneous architectures with these lessons in mind.