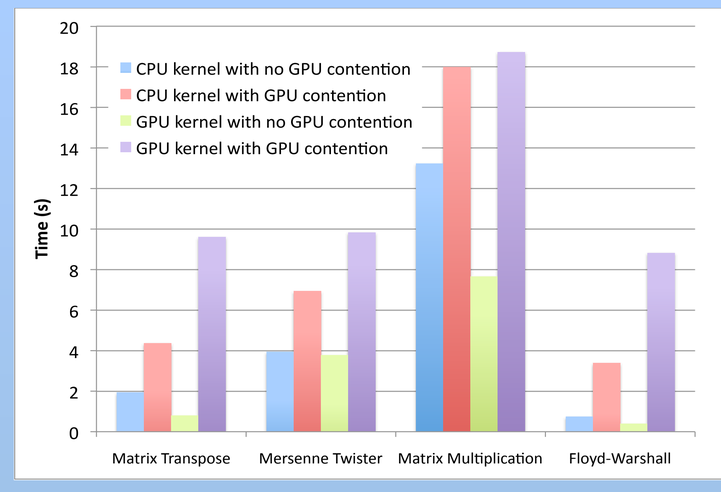
contention-aware scheduling of parallel code for heterogeneous systems

Chris Gregg, Jeff S. Brantley, Kim Hazelwood

Why is contention important?

Scheduling kernels on a heterogeneous system needs to take into consideration whether a device is currently busy executing code because launching a kernel on a busy device can lead to bottlenecks and idle processing. The goal is to formulate a scheduling algorithm that uses contention to minimize the run time for each application, whether it runs on the GPU or the CPU.



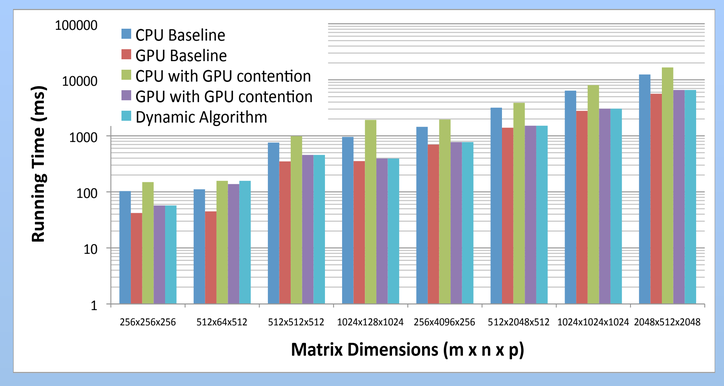
What information is available to make a scheduling decision?

- Whether or not the device is busy
- Input data size for the kernel
- Baseline running times
- Historical running times

Depending on input size, a kernel may run faster on the CPU than the GPU

Application	Faster Processor (Baseline)
Binary Search	Depends
Embarrassingly Parallel	GPU
Matrix Multiply	GPU
Matrix Transpose	GPU
Mersenne Twister	Depends

```
int chooseDevice(int inputDimensions[],bool dataInCache,
                 float choiceOverheadTime) {
  int useGpu = 1;
  int useCpu = 2;
  int contentionFactor = 2;

  GpuBaselineTime = getHistoricalGpuBaseline(inputDimensions);
  CpuBaselineTime = getHistoricalCpuBaseline(inputDimensions);

  if (CpuBaselineTime < choiceOverheadTime) return useCPU;

  bool cpuBusy = checkIfCpuBusy();
  bool gpuBusy = checkIfGpuBusy();

  if ((gpuBusy && cpuBusy) || (!gpuBusy && !cpuBusy))
     if (GpuBaselineTime < CpuBaselineTime) return useGPU;
     else return useCPU;
  else if (gpuBusy && (GpuBaselineTime *
                       contentionFactor < CpuBaselineTime)
     return useGPU;
  else if (gpuBusy && (GpuBaselineTime *
                       contentionFactor >= CpuBaselineTime)
     return useCPU;
  else if (cpuBusy && (CpuBaselineTime *
                       contentionFactor < GpuBaselineTime)
     return useCPU;
  else return useGPU;
}

void updateHistory(deviceType, inputDimensions[],
                   newKernelTime){
  readHistoryFromFile(deviceType,inputDimensions[],
                      &historicalTime,&historyCounter);
  historicalTime = (historicalTime * historyCounter
                    + newKernelTime) / ++historyCounter;
  writeHistoryToFile(deviceType,inputDimensions[],
                     historicalTime,historyCounter);
}
```

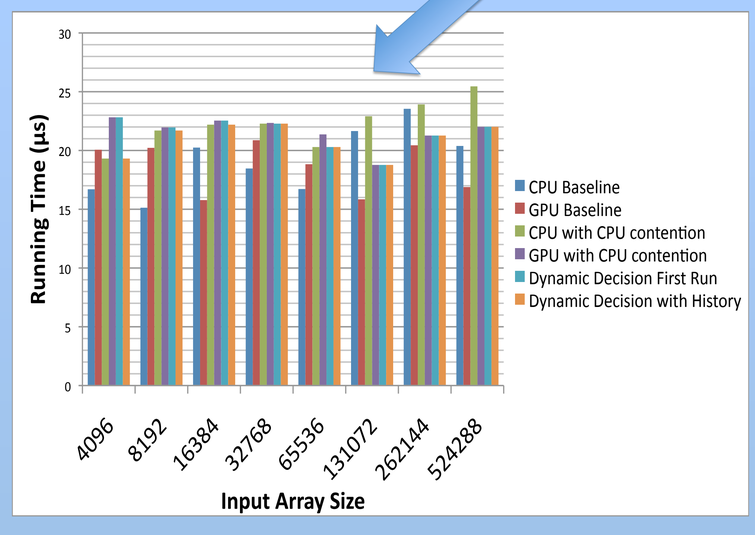


What about data transfer time?

Binary Search results with CPU under contention. For small values of input array size, the GPU is hindered by data transfer onto the device. The dynamic algorithm did not choose correctly for the lowest three input array sizes on the first run, but on the second run the algorithm used the history to correctly choose the best device.

Future Work

An intelligent runtime decision, particularly with regard to contention, would require knowledge of other active kernels' expected execution times. Future work might focus on a more centralized, common kernel scheduler to achieve the best performance for the whole set of active kernels. In addition to performing the decision-making algorithm in the runtime environment, it would would also be beneficial if the operating system (with more contention data available) takes part in the scheduling decision. Additionally, the sample applications were tested with an input size up to the memory limit of the GPU. Yet applications with larger data sets could be refactored to use multiple kernel calls, incurring with associated overheads of transferring data between main memory and the GPU. Future work will explore the effects on performance of splitting a task over multiple kernel calls.



University of Virginia
School of Engineering and Applied Science