DYNAMIC KERNEL SCHEDULER (DKS) – ACCELERATING THE OBJECT ORIENTED PARTICLE ACCELERATOR LIBRARY (OPAL)

U. Locāns, University of Latvia, Riga, Latvia and PSI, Villigen, Switzerland
A. Adelmann, A. Suter, PSI, Villigen, Switzerland

Abstract

Hardware accelerators, such as graphical processing units (GPU) and Intel Many Integrated Core (MIC) processors, provide a huge performance potential for HPC applications. However, due to different hardware architectures and development frameworks, developing fast and manageable code for these devices is a challenging task, especially when integrating the usage of co-processors in large applications, such as OPAL. DKS provides a slim software layer that separates device specific code from the host application and provides a simple interface to communicate and schedule task execution on the device. Algorithms in DKS are implemented using CUDA, OpenCL, and OpenMP to target different devices. The first version of DKS was integrated in OPAL to speed up the FFT based Poisson solver and Monte Carlo simulations for particle matter interaction. The concepts of DKS and its integration with OPAL will be presented, as well as results showing speedups in the range of 9x to 150x for OPAL simulations using GPU or Intel MIC.

INTRODUCTION

Dynamic Kernel Scheduler (DKS) and its integration in Object Oriented Particle Accelerator Library (OPAL) is presented in this work. The aim of DKS is to ease the use of hardware accelerators in large host applications, such as OPAL. This is achieved by seperating all the accelerator and framework specific code in a seperate layer and providing a simple interface that can be implemented in the host application. This interface allows host application to communicate with DKS to schedule memory management, data transfer and kernel execution on the device. DKS contains function implementations writeen using CUDA, OpenCL, and OpenMP to allow the targeting of different accelerators that may be available on the system.

The ability of DKS to have implementations using different frameworks and libraries, and switch between them from the host applications allows not only to target hardware accelerators of different types and fine tune code to gain the maximum performance from each device, but also provides some software investment protection. In case some hardware architecture is no longer manufactured or some new architecture or development framework emerges only DKS needs to be updated.

The first version DKS was integrated into OPAL to accelerate its FFT based Poisson solver and Monte Carlo simulations. This DKS version uses CUDA kernels and OpenMP offload pragms to allow OPAL to accelerate the code using GPU or Intel Many Integrated Core devices.

DKS CONCEPT AND ARCHITECTURE

DKS is a slim software layer between the host application and the hardware accelerator, as depicted in Figure 1. The aim of the DKS is to allow the creation of fast fine tuned kernels using device specific frameworks such as CUDA, OpenCL, and OpenMP. On top of that, DKS allows the easy use of these kernels in host applications without providing any device or framework specific details. This approach facilitates the integration of different types of devices in the existing applications with minimal code changes and makes the device and the host code a lot more manageable.

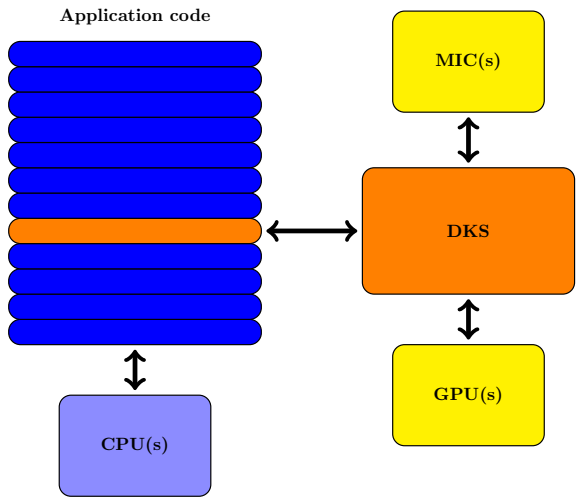

Figure 1: The concept of the Dynamic Kernel Scheduler.

DKS architecture is split into three seperate parts:

1. Communication functions, that handle memory allocation and data transfer to, and from, the device. All the memory management is left up to the user so the data transfers and memory allocation can be scheduled only when necessary. GPU streams are suported so asynchronous data transfer and kernel execution can be implemented.
2. Function library, witch contains algorithms written in CUDA, OpenCL and OpenMP to target different devices and DKS can switch between implementations based on the hardware that is available.
3. Auto-tuning functionality, which is not part of the first DKS version. The aim of auto-tuning is to select the appropriate implementation of the algorithm and change the launch parameters according to the devices that are available on the system in order to gain the maximum performance.

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-136-6

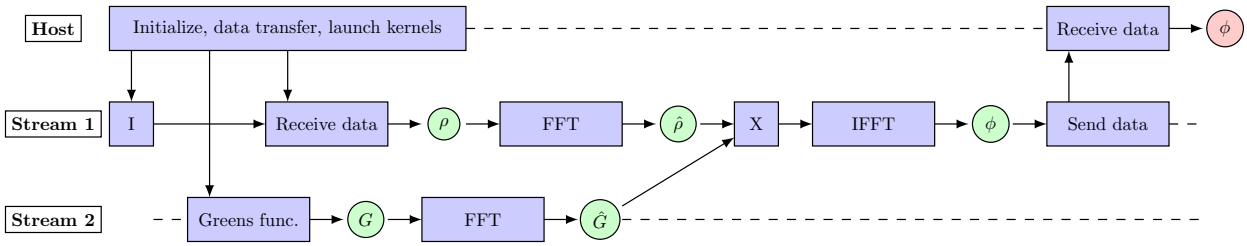

Figure 2: Sequence diagram of the FFT Poisson Solver run with DKS.

OPAL AND DKS

OPAL (Object Oriented Particle Accelerator Library) is a parallel, open source C++ framework for general particle accelerator simulations which includes 3D space charge, short range wake fields, and particle matter interaction. OPAL is based on IPPL (Independent Parallel Particle Layer) which adds parallel capabilities. Main functions inherited from IPPL are structured rectangular grids, fields, parallel FFT, and particles with the respective interpolation operators. Other features are expression templates, and massive parallelism (up to 65000 processors) which allows it to tackle the largest problems in the field.

FFT Poisson Solver in OPAL

The Poisson solver is an essential part of any self-consistent electrostatic beam dynamics code. From the – time to solution – point of view, we observe that in the order of 1/3 of the computational time is spend in this algorithm.

In many of the physics application, the bunch can be considered as small compared to the transverse size of the surrounding beam pipe ($\partial\Omega$). If this is the case the conducting walls can be neglected and, we can solve an open boundary problem. Here we follow the method of Hockney and compute the potential on a grid of size $2^3 M_x M_y M_z$, assuming 3 spatial dimensions of the problem. The calculation then is making use of Fast Fourier Transform (FFT) techniques, with a computational effort scaling as $O(2^3 M_x M_y M_z (\log_2 2 M_x M_y M_z)^3)$ [1–3].

FFT Poisson Solver in DKS

In DKS the FFT Poisson solver is implemented using CUDA to target NVIDIA's GPUs. It uses NVIDIA's cuFFT library to perform the FFT, separate kernels to calculate the Greens function and perform the multiplication on the GPU. To overlap the computation of the Green's function and the transfer of the charge density, ρ, CUDA streams are used. The implementation in DKS allows for multiple CPUs on the same node to share one GPU device, this is achieved using CUDA inter process communications. For the FFT Poisson solver one of the MPI processes acts as a main process and initializes memory allocation and kernel execution on the device. The other MPI processes meanwhile only send and receive data to and from the GPU. The sequence diagram in Figure 2 shows the steps executed for the FFT Poisson solver on the host and GPU.

Particle Matter Interaction in OPAL

One of the features in OPAL is the ability to perform Monte Carlo simulations of the particle beam interaction with matter. This is mostly used for simulations of degrader for proton therapy. A fast charged particle moving through the material undergoes collisions with the atomic electrons and loses energy. In addition, particles are also deflected from their original trajectory due to the Coulomb scattering with nuclei, as shown in Figure 3. The energy loss in OPAL is calculated using Bethe-Bloch formula and the change of particle trajectory is simulated using Multiple Coulomb Scattering and Single Rutherford Scattering [4–6].

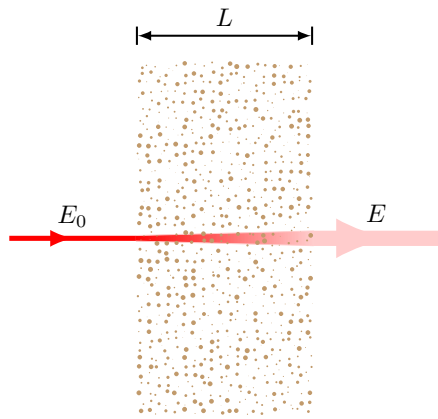

Figure 3: Particle matter interaction, final energy $E < E_0$ and larger momenta spread due to Coulomb scattering and the large angle Rutherford scattering.

At every time step when the particle beam is inside a material the following steps are executed.

- calculate the energy loss of the beam
- delete the particle if the particle's kinetic energy is smaller than a given threshold
- apply Coulomb & Rutherford scattering to the beam

Particle Matter Interaction in DKS

DKS contains CUDA and OpenMP implementations allowing OPAL to offload the simulations of particle matter interactions to GPUs and Intel MIC devices. DKS contains algorithms to allow offload the simulations of energy loss, Coulomb scattering and Rutherformd scattering. In adition particle drift before and after the material using time integration scheme can also be offload to the device.

ISBN 978-3-95450-136-6

Copyright © 2015 CC-BY-3.0 and by the respective authors

Communication between host and device is minimized as much as possible by keeping the particles that are moving through the material in the device memory. They are only transfered to the host once they exit the material. To get the correct distributions for energy loss and scattering NVIDIAS's cuRand and Intels MKL VSL libraries are used. In order to improve the data access paterns of the algorithm and assist with the vectrorization for the Intel MIC, data layout of the particles is organized in structure of arrays format.

RESULTS

To test OPAL with DKS a series of test simulations where run. First the simulations where run on the CPU, where original OPAL's implementations of FFT Poisson solver and Monte Carlo simulations are used. Additional test runs where OPAL uses DKS to offload these tasks to GPU or Intel MIC where perfomed to test the performance of the algorithms in DKS. The test system consists of host with two Intel Xeon e5-2609 v2 processors with total 8 CPU cores, a Nvidia Tesla K20, Nvidia Tesla K40 and a Intel Xeon Phi 5110p.

To test the OPAL FFT Poisson solver performance with DKS, a similar problem setup as reported in [7, 8] was used. Results in Table 1 show the results from CPU simulations run on 8 cores and simulations using GPU. For larger grid sizes where the calculation of Greens function can be overlaped with data transfer we can observe a speedup of up to ×12 compared to the CPU version.

Table 1: OPAL FFT Poisson Solver Results

FFT size	DKS	FFTPoisson Time (s)	FFTPoisson speedup
64×64×32	no	22.53	
	K20	**7.42**	**×3**
	K40	**7.32**	**×3**
128×128×64	no	206.73	
	K20	**32.15**	**×6.5**
	K40	**25.87**	**×8**
256×256×128	no	1879.84	
	K20	**202.63**	**×9.3**
	K40	**160.87**	**×11.7**

To test the Monte Carlo simulations for particle matter interaction a test case where particle beam moves through a $L = 1$ cm thick graphite slab, mimicking a degrader device used in proton therapy. Table 2 shows the benchmark results for these simulations using varios number of particles. Due to current limitations of OPAL CPU simulations where run only on one CPU core. The results show that significant speedup in these simulations can be achieved on the accelerators compared to the host, but it is also observed that GPU significanlty outperforms Intel MIC.

Table 2: OPAL Degrader Results

Particles	DKS	Degrader time(s)	Degrader speedup	Integration time(s)	Integration speedup
10^5	no	20.30		3.46	
	MIC	**2.29**	**×8**	**0.89**	**×4**
	K20	**0.28**	**×72**	**0.15**	**×23**
	K40	**0.19**	**×107**	**0.14**	**×24**
10^6	no	206.77		34.93	
	MIC	**5.38**	**×38**	**4.62**	**×7.5**
	K20	**1.41**	**×146**	**1.83**	**×19**
	K40	**1.18**	**×175**	**1.21**	**×29**
10^7	no	2048.25		351.64	
	K20	**14.4**	**×142**	**17.21**	**×20**
	K40	**12.79**	**×160**	**11.43**	**×30**

SUMMARY

In this paper we presented the first version of Dynamic Kernel Scheduler whitch provides a software layer between host application and hardware accelerators. This allows to create a fine tuned code for different hardware accelerators using different frameworks and easily integrate it into existing host applications. DKS was integrated into OPAL to offload FFT based Poisson solver and Monte-Carlo simulations for particle matter interaction to GPU and Intel MIC using either CUDA or OpenMP. The results of this work show that DKS can be used to substantially speed up existing host applications with minimal additions and changes to host code. Seperating the device specific code in a different layer allows managing and fine tunining the code more easily and it also keeps the host application a lot more portable since all the device and framework specific details are handled by DKS.

REFERENCES

[1] R.W. Hockney, Methods Comput. Phys. 9, 136-210 (1970).

[2] J.W. Eastwood and D.R.K. Brownrigg, J. Comp. Phys, 32, 24-38 (1979).

[3] R.W. Hockney and J.W. Eastwood, "Computer Simulation using Particles," Taylor & Francis Group (1988).

[4] *Stopping Powers and Ranges for Protons and Alpha Particles*, ICRU Report 49 (1993).

[5] K.A. Olive et al., *Particle Data Group*, Chin. Phys. C, 38, 090001 (2014).

[6] W.R. Leo, *Techniques For Nuclear And Particle Physics Experiments*.

[7] Y. Bi, A. Adelmann, et al., Phys. Rev. STAB **14**(5) 054402 (2011).

[8] J. Yang, Adelmann, et al., Phys. Rev. STAB **13**(6) 064201 (2010).

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-136-6