Auto-generating optimized CUDA for stencil computation

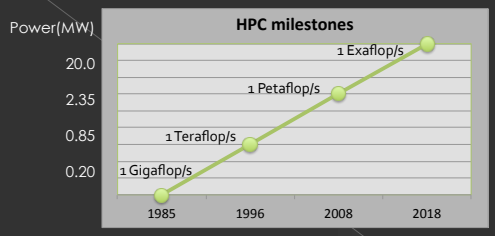
Mint Programming Model

Didem Unat, Xing Cai, Scott B. Baden
dunat@lbl.gov

Lawrence Berkeley National Laboratory
Simula Research Laboratory
University of California, San Diego

Oslo, Jun 07, 2012

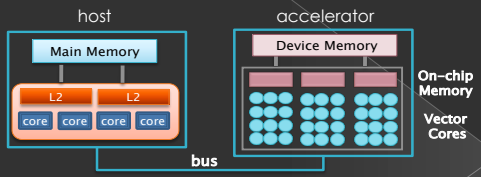
The March to Exascale



- Power consumption is the main design constraint
- Drastic changes in node architecture [Shalf, VecPar'10]
- More parallelism on the chip
- Software-managed memory / incoherent caches
- Already started seeing concrete instances

Accelerator-based Systems

- Graphics Processing Units (GPUs)
  - Massively parallel single chip processor
  - Low power cores: trade off single thread performance
  - Large register file and software-managed memory
- Effective in accelerating certain data parallel applications
  - Case Study: Cardiac Electrophysiology [Unat, PARA'10]
  - Not ideal for others: sorting [Lee, ISCA'10]



Stencil Computations

- Important class of applications (one of the motifs)
  - Basis for approximating derivatives numerically
  - Physical simulations (e.g. turbulence flow, seismic wave propagation)
  - Multimedia applications (e.g. image smoothing)
  - Nearest neighbor update on a structured grid

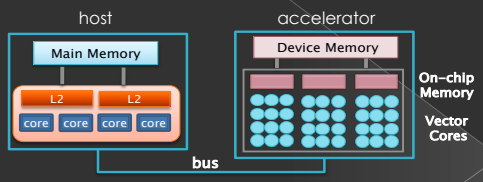
3D Heat Eqn using
fully explicit finite differencing: $\partial u/\partial t = \kappa \nabla^2 u$

```
U'(x,y,z) = c0*U(x,y,z) + c1*(U(x,y,z-1) + U(x,y,z+1)+ U(x,y-1,z)
                               + U(x,y+1,z) + U(x-1,y,z) + U(x+1,y,z))
```

- Highly data parallel, memory bandwidth bound
  - GPU speedups over multicore (8 cores)
    - 5X for Lattice Boltzmann [Lee,ISCA'11],
    - 4X Reverse Time Migration [Kruger, SC'11]

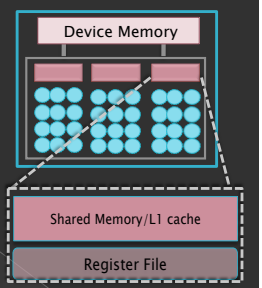
Programming Challenges

- Heterogeneity in compute resources
- Explicit management of data transfer
  - Separate device memory from the host memory
- Reengineering of scientific applications
  - Algorithmic changes to match the hardware capabilities
  - Best performance requires non-trivial knowledge of the architecture



Programming Challenges (cont'd)

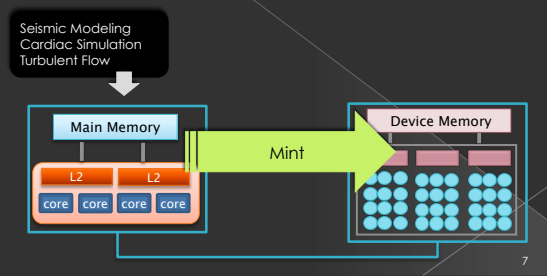
- Explicitly managed memory
  - On-chip memory resources
  - Private and incoherent
  - e.g __shared__ float A[N];
- Hierarchical thread management
  - Thread, thread groups, thread subgroups
  - Granularity of a thread
- Domain-specific optimizations
- Limits the adoption in scientific computing



We need programming models to master the new technology and make it accessible to computational scientists.

Mint Programming Model

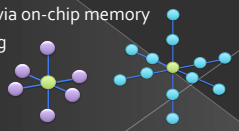
- Aims programmer's productivity and high performance
- Simplifies application development
- Based on a modest number of compiler directives
  - #pragma mint for
  - Incremental parallelization
- Abstracts away the programmer's view of the hardware



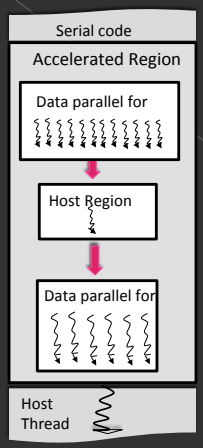
Mint Translator and Optimizer

- Source-to-source translator for the Nvidia GPUs
  - Parallelizes loop nests
  - Relieves the programmer of a variety of tedious tasks

- Motif-specific auto-optimizer
  - Targets stencil methods
  - Incorporates semantic knowledge to compiler analysis
  - Performs data locality optimizations via on-chip memory
  - Compiler flags for performance tuning

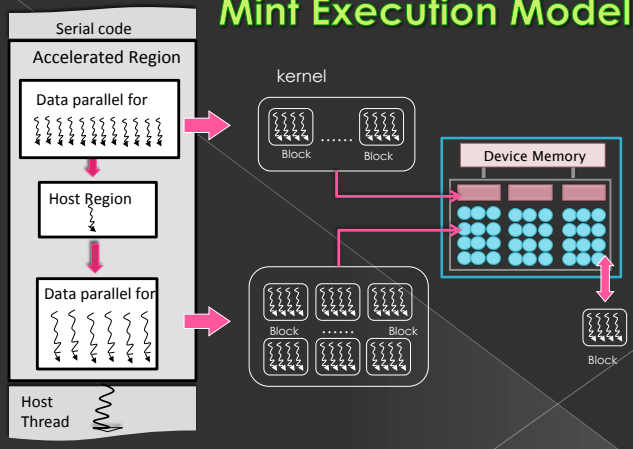


Mint Execution Model



Mint Interface

Mint Execution Model



Mint Interface

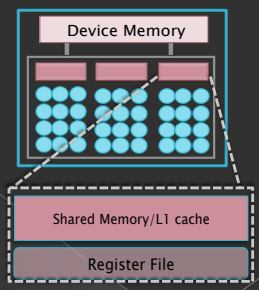
- #pragma mint parallel — Accelerated Region
  - Indicates the accelerated region
- #pragma mint for
  - Marks enclosed loop-nest for acceleration
  - 3 additional clauses for optimizations
- #pragma mint copy — Data Transfer
  - Expresses data transfers between the host and device
- #pragma mint single
  - Handles serial section
- #pragma mint barrier
  - Synchronizes host and device threads

Synchronization

Mint Interface (cont'd)

- Performance tuning parameters
- High-level interface to low-level hardware specific optimizations

1. For-loop clauses
   - handle data decomposition and thread management
   - nest( ), tile( ), chunksize( )
2. Compiler flags for data locality
   - Register: -register
   - Software-managed memory: -shared
   - Cache: -preferL1



Mint Interface

```
#pragma mint copy(U,toDevice,(n+2),(m+2),(k+2))
#pragma mint copy(Unew,toDevice,(n+2),(m+2),(k+2))

#pragma mint parallel
{
  while( t++ < T ){
  #pragma mint for nest(all) tile(16,16,64) chunksize(1,1,64)
   for (int z=1; z<= k; z++)
    for (int y=1; y<= m; y++)
     for (int x=1; x<= n; x++)
        Unew[z][y][x] = c0 *  U[z][y][x] +
                        c1 * (U[z][y][x-1] + U[z][y][x+1] +
                        U[z][y-1][x] + U[z][y+1][x] +
                        U[z-1][y][x] + U[z+1][y][x]);
   double*** tmp;
   tmp = U; U = Unew; Unew = tmp;

 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```

Mint Program for the 3D Heat Eqn.

Mint Interface

```
#pragma mint copy(U,toDevice,(n+2),(m+2),(k+2))
#pragma mint copy(Unew,toDevice,(n+2),(m+2),(k+2))

#pragma mint parallel
{
  #pragma mint for
    for (int z=1; z<= k; z++)
     for (int y=1; y<= m; y++)
      for (int x=1; x<= n; x++)
         Unew[z][y][x] = c0 *  U[z][y][x] +
                         c1 * (U[z][y][x-1] + U[z][y][x+1] +
                         U[z][y-1][x] + U[z][y+1][x] +
                         U[z-1][y][x] + U[z+1][y][x]);
   double*** tmp;
   tmp = U; U = Unew; Unew = tmp;

 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```

Data parallel for loop

Mint Interface

```
#pragma mint copy(U,toDevice,(n+2),(m+2),(k+2))
#pragma mint copy(Unew,toDevice,(n+2),(m+2),(k+2))

#pragma mint parallel
{
  while( t++ < T ){
  #pragma mint for nest(all) tile(16,16,64) chunksize(1,1,64)
   for (int z=1; z<= k; z++)
    for (int y=1; y<= m; y++)
     for (int x=1; x<= n; x++)
        Unew[z][y][x] = c0 *  U[z][y][x] +
                        c1 * (U[z][y][x-1] + U[z][y][x+1] +
                        U[z][y-1][x] + U[z][y+1][x] +
                        U[z-1][y][x] + U[z+1][y][x]);
   double*** tmp;
   tmp = U; U = Unew; Unew = tmp;

 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```

Accelerated Region

Mint Interface

```
#pragma mint copy(U,toDevice,(n+2),(m+2),(k+2))
#pragma mint copy(Unew,toDevice,(n+2),(m+2),(k+2))

#pragma mint parallel
{
  #pragma mint for nest(all)
    for (int z=1; z<= k; z++)
     for (int y=1; y<= m; y++)
      for (int x=1; x<= n; x++)
         Unew[z][y][x] = c0 *  U[z][y][x] +
                         c1 * (U[z][y][x-1] + U[z][y][x+1] +
                         U[z][y-1][x] + U[z][y+1][x] +
                         U[z-1][y][x] + U[z+1][y][x]);
   double*** tmp;
   tmp = U; U = Unew; Unew = tmp;

 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```

depth of loop parallelism

Mint Interface

depth of loop parallelism

partitioning iteration space

```
#pragma mint for nest(all) tile(16,16,64)
    for (int z=1; z<= k; z++)
     for (int y=1; y<= m; y++)
      for (int x=1; x<= n; x++)
         Unew[z][y][x] = c0 *  U[z][y][x] +
                         c1 * (U[z][y][x-1] + U[z][y][x+1] +
                         U[z][y-1][x] + U[z][y+1][x] +
                         U[z-1][y][x] + U[z+1][y][x]);
    tmp = U; U = Unew; Unew = tmp;
 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```


Mint Compiler

Mint Interface

depth of loop parallelism

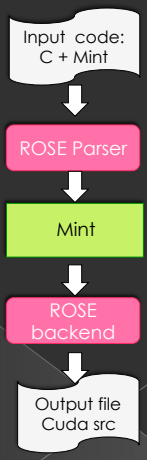
partitioning iteration space

workload of a thread

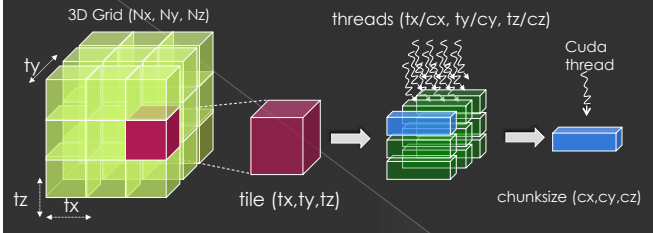
```
#pragma mint for nest(all) tile(16,16,64) chunksize(1,1,64)
    for (int z=1; z<= k; z++)
     for (int y=1; y<= m; y++)
      for (int x=1; x<= n; x++)
         Unew[z][y][x] = c0 *  U[z][y][x] +
                         c1 * (U[z][y][x-1] + U[z][y][x+1] +
                         U[z][y-1][x] + U[z][y+1][x] +
                         U[z-1][y][x] + U[z+1][y][x]);
    tmp = U; U = Unew; Unew = tmp;
 }//end of while
}//end of parallel region
#pragma mint copy(U,fromDevice,(n+2),(m+2),(k+2))
```


Mint Compiler

- Fully automated translation and optimization system
  - Transformation performed on the Abstract Syntax Tree
- Built on top of the ROSE compiler
  - Developed at LLNL
  - ROSE provides an API for generating and manipulating Abstract Syntax Trees
- Mint is a part of the ROSE distribution since Nov'11.

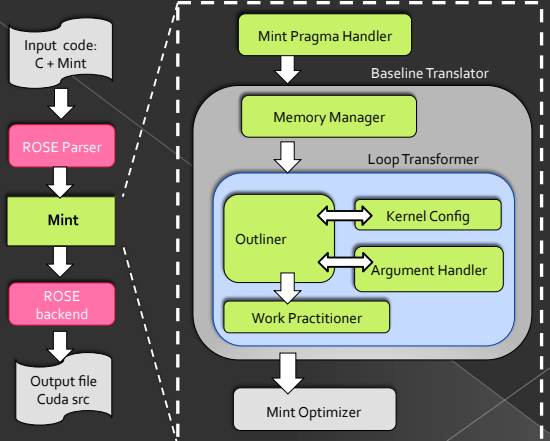


mint for [clauses]



`#pragma mint for nest(#,all) tile(`t_x`,`t_y`,`t_z`) chunksize(`c_x`,`c_y`,`c_z`)`

Manages data decomposition and thread work-assignment.



Outlining

```
#pragma mint parallel
{
  while(t < T)
  {
    t += dt;

    #pragma mint for
    for (i=1 ; i <= N ; i++)
      for(j=1 ; j <= N ; j++)
        A[i][j] = c*(B[i-1][j] + B[i+1][j]
                  +(B[i][j-1] + B[i][j+1]);

  }//end of while
}//end of parallel region
```


Outlining

```
#pragma mint parallel
{
  while(t < T)
  {
    t += dt;

    #pragma mint for
    for (i=1 ; i <= N ; i++)
      for(j=1 ; j <= N ; j++)
        A[i][j] = c*(B[i-1][j] + B[i+1][j]
                  +(B[i][j-1] + B[i][j+1]);

  }//end of while
}//end of parallel region
```

Host

```
/* Outlined Kernel */
__global__ void cuda_func(…)
{
   . . .
}
```

Device

Outlining

```
#pragma mint parallel
{
  while(t < T)
  {
    t += dt;

     . . .
    cuda_1_func_<<<threads, blocks>>>
   (…);
     . . .

  }//end of while
}//end of parallel region
```

Host

```
/* Outlined Kernel */
__global__ void cuda_func(…)
{
   . . .
}
```

Device

Minted Kernel Code

```
__global__ void
mint_1_1517(cudaPitchedPtr ptr_dU ...)
{
  double* U = (double *)(ptr_dU.ptr);
  int widthU = ptr_dU.pitch / sizeof(double );
  int sliceU = ptr_dU.ysize * widthU;

  ...

  int _idx = threadIdx.x + 1;
  int _gidx = _idx + blockDim.x * blockIdx.x;

  ...

  if (_gidz >= 1 && _gidz <= k)
   if (_gidy >= 1 && _gidy <= m)
    if (_gidx >= 1 && _gidx <= n)

       Unew[indUnew] = c0 * U[indU]
         + c1 * (U[indU - 1] + U[indU + 1]
                  . . . );

}//end of kernel
```

Unpack CUDA pitched ptrs

Compute local and global indices using thread and block IDs

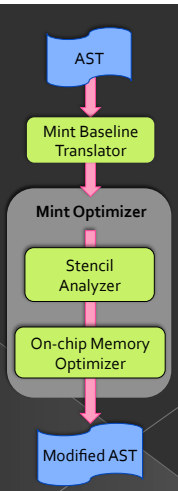
If-statements are derived from for-statements

Each CUDA thread updates single data point

Mint Optimizer

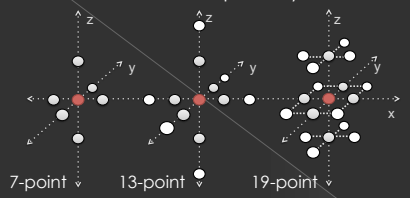
Mint Optimizer

- The baseline translator
  - performs all the memory references through global memory
  - can still parallelize loops, launch kernel, perform data transfers
- Optimizer focuses on stencil methods
  - Reduces global memory accesses
  - Data locality using on-chip memory
- On-chip memory optimization flags
  - -preferL1, -register, -shared
  - Best performance depends on the device and application

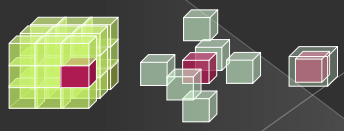


Stencil Analyzer

- Analyzes array access pattern
  - Finds stencil structure and dependency between threads



- Ghost cell region

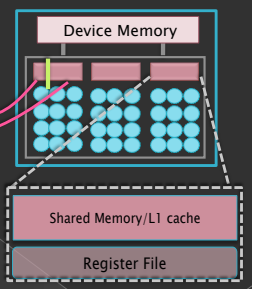


Software-managed Memory Optimizer

- Trade off: find the sweet spot
- Which variables? and How many variables ?
  - Maximize the total reduction in global memory references
  - Minimize the shared memory usage
  - Planes may come from 1 or more arrays

On-chip Memory Optimizer

- -preferL1
  - Configures on-chip memory on Fermi
  - Favors 48KB L1 and 16KB shared memory

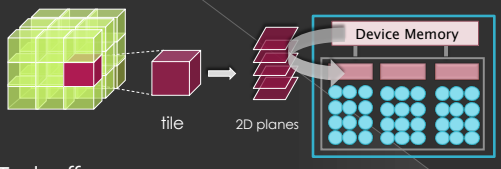


- -register
  - Takes advantage of large register file
  - Places frequently accessed arrays into registers
  - Enhances access to the central point of a stencil

Performance Results

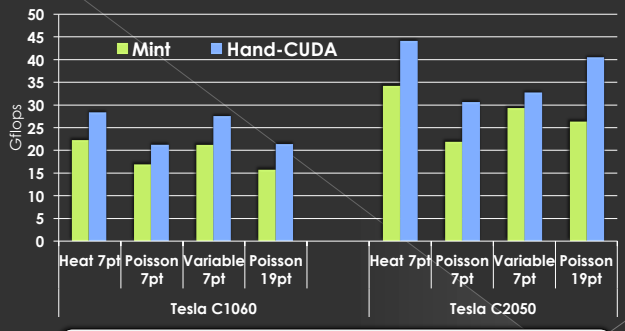
Software-managed Memory Optimizer

- -shared
  - Detects sharable references among threads
  - Places them in shared memory (software-managed memory)
  - A number of planes reside on shared memory



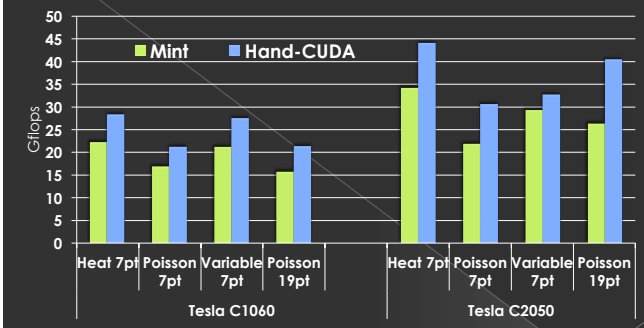
- Trade off
  - Reduces memory references to frequently accessed locations
  - Increases resources needed by a thread, reducing concurrency

Mint is competitive with hand-coding



D.Unat, X.Cai, and S. Baden. *"Mint: Realizing CUDA performance in 3D Stencil Methods with Annotated C", in International Conference on Supercomputing. ICS'11, Tucson, AZ, 2011*

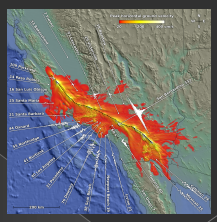
Mint is competitive with hand-coding



On Tesla C1060, Mint achieves 79% of the hand-optimized CUDA.
On Tesla C2050 (Fermi), Mint achieves 76% of the hand-optimized CUDA.

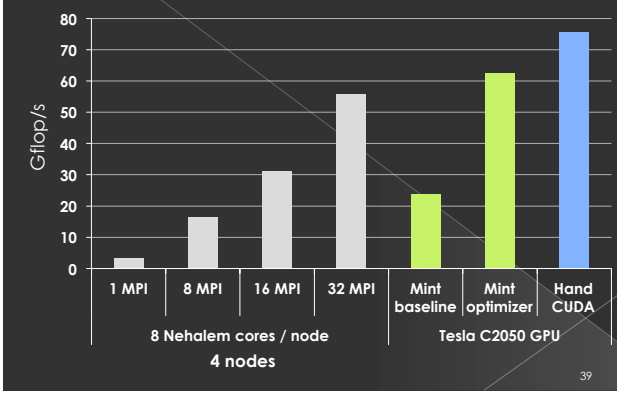
AWP-ODC Earthquake Modeling

- Petascale anelastic wave propagation code
  - Used by researchers Southern CA Earthquake Center
  - Earthquake-induced seismic wave propagation
- Gordon Bell Prize finalist at SC'10
  - Yifeng Cui and Jun Zhou at SDSC
- Refers to 31 three-dim arrays
  - asymmetric 13-point stencil
- Time consuming loops: 185 lines
- Generated CUDA code: 1185 lines



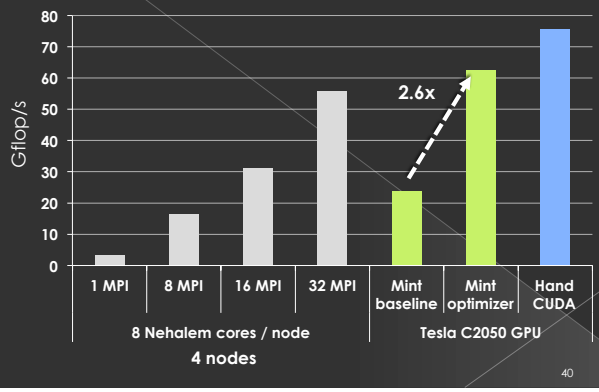
D. Unat, J. Zhou, Y. Cui, X. Cai, and S. Baden. "*Accelerating a 3D Finite Difference Earthquake Simulation with a C-to-CUDA Translator*", in Computing in Science and Engineering Journal, 2012.

AWP-ODC Results



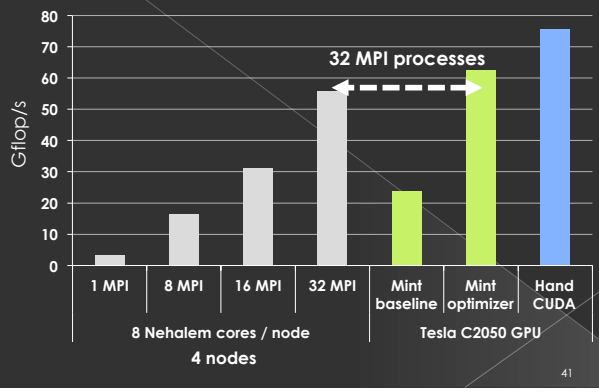
AWP-ODC Results

The Mint optimizer improves the performance 2.6x over the Mint baseline.



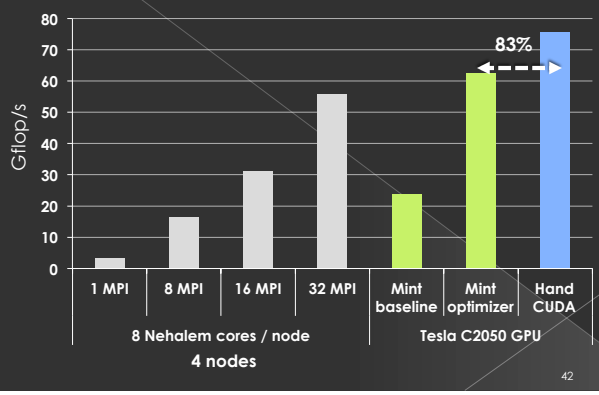
AWP-ODC Results

The Minted code on single GPU is slightly faster than 32 cores.



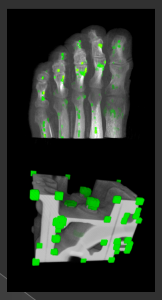
AWP-ODC Results

The Minted code achieves 83% of the hand-optimized CUDA.



Harris Interest Point Algorithm

- Computer vision algorithm
  - Detects corners and high intensity points
  - Collaborated with Han Kim & Jürgen Schulze
- Inserted 5 lines of Mint code into the original code (~350 lines)
- Real-time performance with Mint
  - 10-22x performance of 8 Nehalem cores running OpenMP threads
  - Tesla C2050 vs Intel Xeon E5504 Nehalem



D.Unat, H.S. Kim, J. Schulze, S.B. Baden, *"Auto-optimization of a Feature Selection Algorithm"*, in Emerging Applications and Many-core Architecture, EAMA Workshop co-located with ISCA, San Jose, CA, 2011.

Related Work

- OpenMPC extends OpenMP to support CUDA
  - Parallelizes only outer loop and no shared memory optimization
- Commercial compiler from Portland Group (PGI)
  - General-purpose compiler

Gflops	OpenMPC	PGI	Mint	Hand-CUDA
7pt Heat Eqn.	1.06	9.0	22.2	28.3

- OpenACC
  - Collaborative effort from PGI, Cray, CAPs, Nvidia
  - Shows that directive-based model is a promising approach

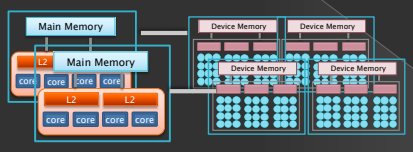
All results are obtained on Tesla C1060. PGI (v11.1) results: on the Lincoln system.

Future Research

- Target multiple GPUs
- MPI code generation
- Extend Mint for Intel Many Integrated Core (MIC)
  - Same Mint execution model
  - New clauses or compiler options
    - Register blocking, SSE instructions, software prefetching

-offload=[mic | apu | cuda]

Maintain single code base?



Conclusion

- Mint Programming Model
  - Addresses the programmability issue of GPUs
    - Today's massively parallel architectures
    - Software-managed storage and massively parallel chip
- Source-to-source translator and optimizer
  - Incorporates motif-specific knowledge
  - Achieved around 80% of the hand-optimized CUDA
    - Both commonly-used kernels and real-world applications
- Available for download
  - Our project website
    - http://sites.google.com/site/mintmodel/
  - Online Translator:
    - http://ege.ucsd.edu/translate.html

Thank you!

Scott Baden (UCSD),
Xing Cai (Simula),
Allan Snavely (SDSC),
Han Suk Kim (UCSD, Apple),
Jürgen Schulze (UCSD),
Yifeng Cui (SDSC),
Jun Zhou (SDSC),
Wenjie Wei (Simula),
Ross Walker (SDSC),
Dan Quinlan (LLNL),
ROSE Team (LLNL),
Paulius Micikevicius (Nvidia),
Everett Phillips (Nvidia)

UCSD

NSF

[simula . research laboratory]

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

References

- [Shalf, VecPar'10] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In Proceedings of the 9th International Conference on High Performance Computing for Computational Science, VECPAR'10.
- [Lee, ISCA'10] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennu- paty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. SIGARCH Comput.
- [Unat, Para'10] Didem Unat, Xing Cai, and Scott Baden. Optimizing the Aliev-Panfilov model of cardiac excitation on heterogeneous systems. Para 2010: State of the Art in Scientific and Parallel Computing
- [Lee] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. OpenMP to GPGPU: a compiler framework for automatic translation and optimization. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '09
- [William] Samuel Webb Williams. Auto-tuning Performance on Multicore Computers. PhD thesis, EECS Department, University of California, Berkeley, Dec 2008
- [Carrington] Laura Carrington, Mustafa M. Tikir, Catherine Olschanowsky, Michael Laurenzano, Joshua Peraza, Allan Snavely, and Stephen Poole. An idiom-finding tool for increasing productivity of accelerators. In Proceedings of the International Conference on Supercomputing, ICS '11
- [Datta] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In SC '08