DFT Transform on the Fermi (GTX480):

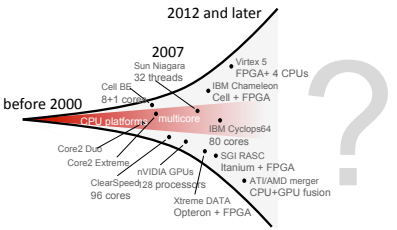
Automatic Program Generation

Christos Angelopoulos, Franz Franchetti and Markus Pueschel

Carnegie Mellon

SPIRAL www.spiral.net

Problem and Goal

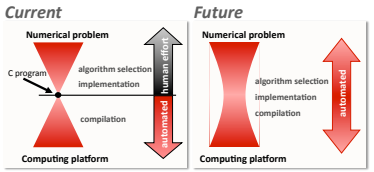


Library development becomes a nightmare

The Complete Picture

Spiral:
Automating high performance library development

Philosophy



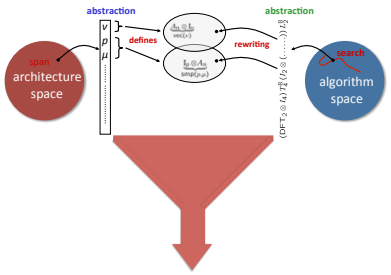
- C code a singularity: Compiler has no access to high level information
- Challenge: conquer the high abstraction level for complete automation

Forward Problem: Match Algorithm to Architecture

Architectural parameter: Vector length, #processors, ...

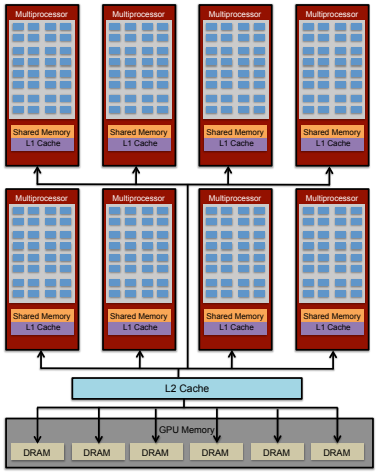
Model: common abstraction = spaces of matching formulas

Transform: problem size, kernel choice




```
a1518 = (2*i528);
s522 = X[a1518];
a1519 = (a1518 + 1);
s523 = X[a1519];
a1520 = (a1518 + 256);
s524 = X[a1520];
a1521 = (a1518 + 257);
s525 = X[a1521];
t1122 = (s522 + s524);
t1123 = (s523 + s525);
t1124 = (s522 - s524);
t1125 = (s523 - s525);
a1522 = (a1518 + 128);
```

GPU Abstraction Model

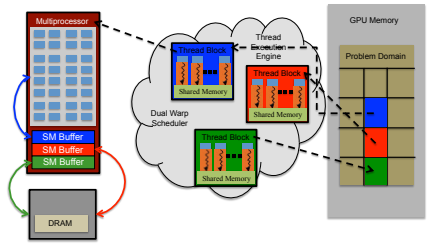


Architecture

- 15 Multiprocessors
- 32 cores per multiprocessor
- 32 K registers per multiprocessor
- 48 KB of shared memory
- 16 KB of L1 cache
- 768 KB of L2 cache
- 1.5 GB of GPU Memory

Restrictions

- Banked Shared Memory
  - 32 banks
    - Within one warp resolve bank conflicts
    - Every thread in the warp Reads/Writes at different bank
    - 32 threads in a warp to 32 banks
- Register pressure
  - Max registers per MP = 32K/# of threads per MP
- Uncommon Architectural Model
  - Size of registers > Size of caches
- Global Memory
  - Only block transfers, using caches
  - Double buffering



Buffering and Programming Model

This work was supported by DARPA DESA Program and Nvidia

References:

1. F. Franchetti, M. Püschel, Y. Voronenko, Sr. Chellappa and J. M. F. Moura
   Discrete Fourier Transform on Multicore
   IEEE Signal Processing Magazine, special issue on ``Signal Processing on Platforms with Multiple Cores", Vol. 26, No. 6, pp. 90-102, 2009
2. M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson and N. Rizzolo
   SPIRAL: Code Generation for DSP Transforms Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation", Vol. 93, No. 2, pp. 232- 275, 2005
3. F. Franchetti, Y. Voronenko and M. Püschel
   FFT Program Generation for Shared Memory: SMP and Multicore Proc. Supercomputing (SC), 2006

Algorithm & Program Generation

Program Generation in Spiral (Sketched)

Transform user specified: DFT_{64}

Fast algorithm in SPL many choices:

$$(\mathrm{DFT}_4 \otimes \mathrm{I}_4)^{(I_4 \otimes L^{16}_4)} \mathrm{D}^{64}_4$$
$$(\mathrm{D}_j \mathrm{DFT}_4 Z^4_j \otimes \mathrm{I}_4)^{(I_4 \otimes L^{16}_4)} (\mathrm{L}^{16}_4 \otimes \mathrm{I}_4)$$
$$(\mathrm{D}^{64}_4)^{(L^{16}_4 \otimes I_4)}$$
$$(\mathrm{D}_j \mathrm{DFT}_4 Z^4_j \otimes \mathrm{I}_4)^{(I_4 \otimes L^{16}_4)} \mathrm{L}^{64}_4$$

Σ-SPL:

$$\sum_j \sum_{j_1} S_{j_1} \mathrm{DFT}_4 \mathrm{diag}(t^4_2 \circ d_j) G_{i_1 j}$$
$$\sum_j \sum_{j_1} S_{j_1} \mathrm{DFT}_4 \mathrm{diag}(t^4_2 \circ d_j) G_{i_1 j, i_2 j}$$
$$\sum_j \sum_{j_1} S_{j_1} \mathrm{DFT}_4 G_{i_1 j, i_2 j}$$

CUDA Code:

```
...
int i17 = (threadIdx.x / 4) % 4;
int i19 = threadIdx.x % 4;
a89 = ((4*i17) + i19);
s41 = s_data[a89];
a90 = (a89 + 64);
s42 = s_data[a90];
a91 = (a89 + 32);
s43 = s_data[a91];
a92 = (a89 + 96);
s44 = s_data[a92];
...
```

Optimization at all abstraction levels:

- parallelization vectorization
- loop optimizations rewriting rules indices and permutations optimizations shifting and twiddles merging
- constant folding scheduling compiler optimizations .....

Iteration of this process to search for the fastest

GPU Stockham Algorithm

Original Stockham :

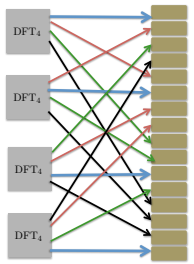
$$\mathrm{DFT}_{r^n} \rightarrow \prod_{i=0}^{n-1} (\mathrm{DFT}_r \otimes \mathrm{I}_{r^{n-1}}) \mathrm{T}^{r^n}_{r^i} (\mathrm{L}^{r^{n-i}}_r \otimes \mathrm{I}_{r^i})$$

GPU Stockham:

$$\mathrm{DFT}_{r^n} \rightarrow \prod_{i=0}^{n-1} (((\mathrm{D}^i_r \cdot \mathrm{DFT}_r \cdot Z^i_q) \otimes \mathrm{I}_\rho \otimes_j \mathrm{I}_\mu \otimes \mathrm{I}_{\frac{n-1}{\rho\mu}}) \mathrm{L}^{r^{n-1}}_r)$$
$$\otimes \mathrm{I}_\rho \otimes \mathrm{I}_\mu \otimes \mathrm{I}_{\frac{1}{\rho\mu}})^{(I_\rho \otimes \mathrm{L}^{\frac{n}{\rho\mu}}_{\rho\mu})(\mathrm{I}_{\frac{n}{\rho\mu}} \otimes L^{\rho\mu}_\rho)} \mathrm{T}^{r^{n-1}}_{r^{n-i}} (\mathrm{L}^{r^{n-1}}_{r^i} \otimes \mathrm{I}_{r^i})$$

Idea :

$$\mathrm{I}_n \otimes \mathrm{DFT}_r = \mathrm{I}_n \otimes_i \mathrm{D}^i_r \cdot \mathrm{DFT}_r \cdot \mathrm{Z}^i_r$$



- uses DFT cyclic shift property
- avoids bank conflicts by adding extra operations
- avoids extra operations by merging twiddle diagonals from different stages

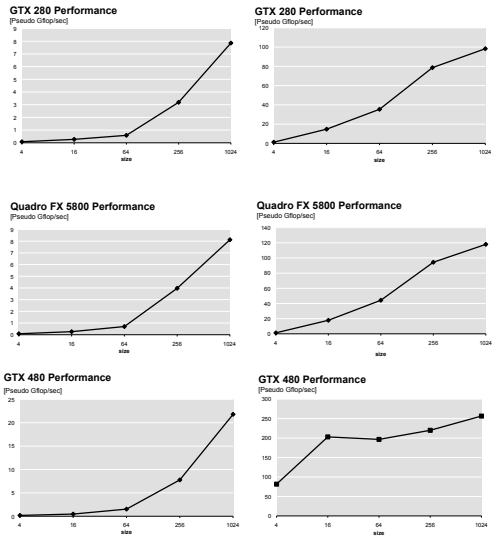
Code Example

```
__global__ void sub(float * X)
{
...
a2101 = (16*i477);
a2102 = ((5*i470) + a2101);
s490 = s_data[a2102];
s491 = s_data[(a2102 + 64)];
b374 = (i470 + a2101);
a2103 = (b374 + (4*((i470 + 2) % 4)));
s492 = s_data[a2103];
s493 = s_data[(a2103 + 64)];
t562 = (s490 + s492);
t563 = (s491 + s493);
t564 = (s490 - s492);
t565 = (s491 - s493);
a2104 = (b374 + (4*((i470 + 1) % 4)));
s494 = s_data[a2104];
s495 = s_data[(a2104 + 64)];
a2105 = (b374 + (4*((i470 + 3) % 4)));
s496 = s_data[a2105];
s497 = s_data[(a2105 + 64)];
t566 = (s494 + s496);
t567 = (s495 + s497);
t568 = (s494 - s496);
t569 = (s495 - s497);
t570 = (t562 + t566);
t571 = (t563 + t567);
t572 = (t562 - t566);
t573 = (t563 - t567);
a2111 = (i470 + (4*i477));
s_data[a2111] = ((texldfetchD60_sv1570) - (texldfetchD60_sv1571));
s_data[(a2111 + 64)] = ((texldfetchD60_sv1570) + (texldfetchD60_sv1571));
s_data[(a2111 + 32)] = ((texldfetchD60_sv1572) - (texldfetchD60_sv1573));
s_data[(a2111 + 96)] = ((texldfetchD60_sv1572) + (texldfetchD60_sv1573));
t574 = (t564 - t569);
t575 = (t565 + t568);
t576 = (t564 + t569);
t577 = (t565 - t568);
s_data[(a2111 + 16)] = ((texldfetchD61_sv1574) - (texldfetchD61_sv1575));
s_data[(a2111 + 80)] = ((texldfetchD61_sv1574) + (texldfetchD61_sv1575));
s_data[(a2111 + 48)] = ((texldfetchD61_sv1576) - (texldfetchD61_sv1577));
s_data[(a2111 + 112)] = ((texldfetchD61_sv1576) + (texldfetchD61_sv1577));
...
}
```

Results

SM to SM Latency

Throughput



Future Work

- N-Dimensional DFTs
- Big (Global Memory) DFTs from GPU Memory to GPU Memory
- Overlapping DFT algorithms from Host to Device
- Work still in progress...