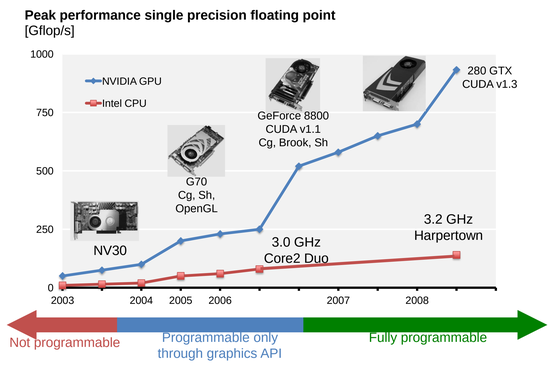
# Automatic Generation of FFT Libraries for GPUs

Christos Angelopoulos, Franz Franchetti and Markus Pueschel

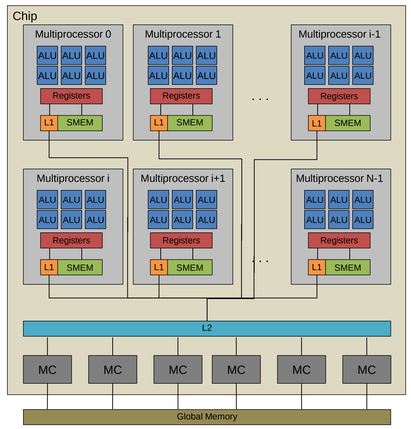
Carnegie Mellon

SPIRAL www.spiral.net

## GPUs and Programmability




## GPU Architecture Model




## Shared Memory Optimized GPU DFT Algorithm

**Original Stockham :**

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

Rewriting

Rules

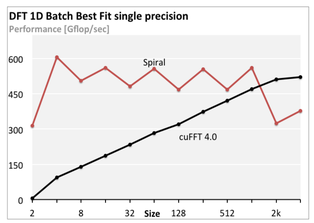
① Loop Splitting
② Loop Interchange
③ Loop Tiling for unit stride outputs
④ Cyclic Shift Property to avoid bank conflicts

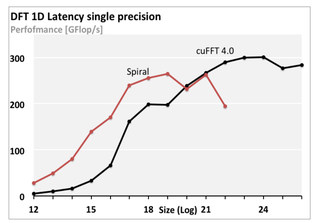
**GPU Stockham :**

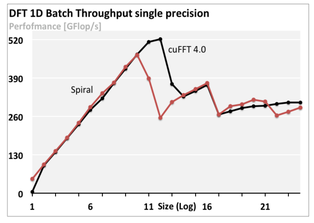
- Inplace algorithm
- Minimize global memory transfers
- Use Shared memory
- Register to register computations

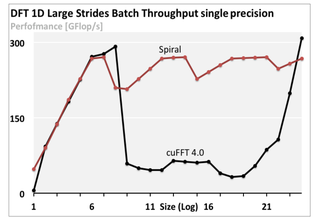
$$\mathrm{DFT}_{r^n} \rightarrow \prod_{i=0}^{n-1} (((\mathrm{D}_r^i \cdot \mathrm{DFT}_r \cdot \mathrm{Z}_4^i) \otimes \mathrm{I}_\rho \otimes_j \mathrm{I}_\mu \otimes \mathrm{I}_{\frac{r^{n-i}}{\rho\mu}}) \mathrm{L}_r^{r^{n-i}}) \otimes \mathrm{I}_\rho \otimes \mathrm{I}_\mu \otimes \mathrm{I}_{\frac{r^i}{\rho\mu}}) (\mathrm{I}_\rho \otimes \mathrm{L}_{\rho\mu}^{\frac{r^n}{\rho}}) (\mathrm{I}_{\frac{r^n}{\rho\mu}} \otimes \mathrm{L}_\rho^{\rho\mu}) \mathrm{T}_{r^{n-i}}^{(\mathrm{L}_r^{r^{n-i}} \otimes \mathrm{I}_{r^i})}$$

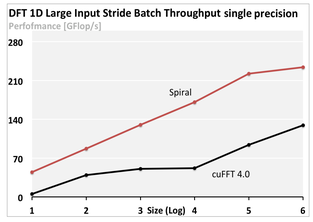
## Results on the GTX 480

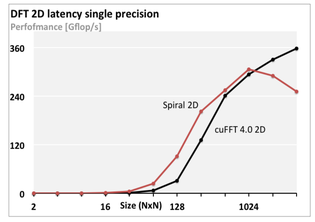



## Philosophy

- **Vendor Libraries**
- **Auto-tuned Libraries**

Problems:
- Expertise required
- Platform specific
- Manual re-tuning
- Extensions to:
  New architectures
  New algorithms

- **Computer Generated Libraries: SPIRAL for GPUs**

Specification → Adaptive GPU library / Platform-tuned code → Auto-tuning at runtime

## 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

- **Key problems**
  - Parallelism, Vectorization, Memory Hierarchy
- **Domain Specific Language**
  $\mathrm{A} \otimes \mathrm{B} = [\alpha_{k,l}\mathrm{B}]$, for $\mathrm{A} = [\alpha_{k,l}]$
- **Address Hardware Characteristics**
- **Advantages**
  - Efficient handling of complexity
  - Efficient porting to new platforms

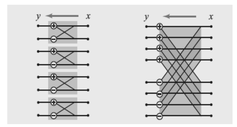
Identify hardware parameters → $vector(v)$

Identify formulas → $\mathrm{A} \otimes \mathrm{I}_v, \quad D_n, \quad \mathrm{L}_v^{v^2}$

Derive algorithm → $\mathrm{DFT}_{mn} = (\mathrm{DFT}_k \otimes \mathrm{I}_m)\mathrm{T}_n^{mn}\mathrm{L}_k^{mn}(\mathrm{DFT}_m \otimes \mathrm{I}_k)$

Generate platform oriented algorithm → $\mathrm{DFT}_{mn} = ((\mathrm{DFT} \otimes \mathrm{I}_{n/v} \otimes \mathrm{I}_v)\mathrm{T}_n^{mn}(\mathrm{I}_{m/v} \otimes (\mathrm{I}_{n/v} \otimes \mathrm{L}_v^{v^2}) (\mathrm{L}_{n/v}^n \otimes \mathrm{I}_v)(\mathrm{DFT}_n \otimes \mathrm{I}_v))(\mathrm{L}_{m/v}^{mn/v} \otimes \mathrm{I}_v)$

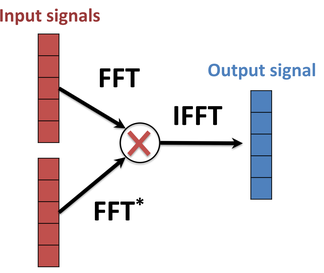
Generate Platform Tuned Code




## Next Step

### Correlation (Frequency Domain)

- **Code generator**
  - Only one data transfer from CPU DRAM to GPU
  - Minimize GPU DRAM memory roundtrips
- **Application Scenario**
  - PDE solvers
    - Huge correlations




## Future Work

- **Fast PDE solvers on GPUs**

This work was supported by DARPA DESA Program and Nvidia

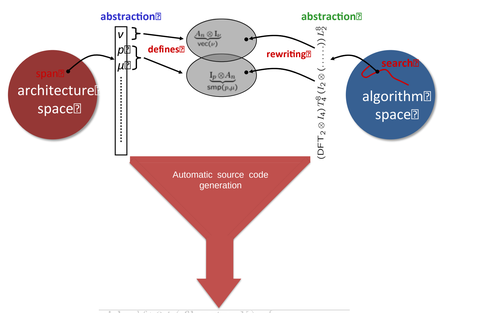
**References:**

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

## Forward Problem: Match Algorithm to Architecture

Platform specification — Problem specification

Architectural parameter: Vector length, # processors, ...
Model: common abstraction = spaces of matching formulas
Transform: problem size, kernel choice




```
void dft64(float* X) {
  a283 = ((20*i10) + i16);
  s97 = s_data[a283];
  ...
  t137 = (s101 + s103);
  t138 = (s102 + s104);
  t139 = (s101 - s103);
  t140 = (s102 - s104);
  ...
  a296 = ((4*i10) + i16);
  s_data[a296] = (t137 + t141);
}
```

Customized, high-performance code for GPU platform
Mapped to target architecture by construction

## Automatic Library Generation With Spiral

### GPU Architectural Constrains in Formulas

Tile for local memory
- Number of Memory Banks
- Length of SIMD unit

Parallelize
- For large sizes re-distribute data
Memory Hierarchy optimizations

Orchestrate permutation shuffles for nicer data accesses

Vectorize packet exchanges

## Algorithm & Program Generation

### GPU Code Through Formula Rewriting

Transform user specified → $\mathrm{DFT}_{64}$

Fast algorithm in SPL choices → $(\mathrm{DFT}_4 \otimes \mathrm{I}_4 \otimes \mathrm{I}_4)^{(I_4 \otimes L_4^{16})} \mathrm{D}_2^{64} (\mathrm{D}_j \mathrm{DFT}_4 \mathrm{Z}_j^4 \otimes \mathrm{I}_4 \otimes_j \mathrm{I}_4)^{(I_4 \otimes L_4^{16})} (\mathrm{L}_4^{16} \otimes \mathrm{I}_4)(\mathrm{D}_1^{64})^{(L_4^{16} \otimes I_4)} (\mathrm{D}_j \mathrm{DFT}_4 \mathrm{Z}_j^4 \otimes \mathrm{I}_4 \otimes_j \mathrm{I}_4)^{(I_4 \otimes L_4^{16})} \mathrm{L}_4^{64}$

∑-SPL → $\sum\sum S_{j,i} \mathrm{DFT}_4 \mathrm{diag}(\mathrm{t}_2^i \circ \mathrm{d}_j) G_{i,j}$
$\sum\sum S_{j,i} \mathrm{DFT}_4 \mathrm{diag}(\mathrm{t}_1^i \circ \mathrm{d}_j) G_{i,j,z_j}$
$\sum\sum S_{j,i} \mathrm{DFT}_4 G_{i,j,z_j}$

CUDA Code →

```
void dft64(float* X) {
  a283 = ((20*i10) + i16);
  s97 = s_data[a283];
  ...
  t137 = (s101 + s103);
  t138 = (s102 + s104);
  t139 = (s101 - s103);
  t140 = (s102 - s104);
  ...
  a296 = ((4*i10) + i16);
  s_data[a296] = (t137 + t141);
}
```

Optimization at all abstraction levels:

- parallelization
  vectorization
  memory architecture optimizations
- loop optimizations
  rewriting rules
  software pipelining
  warp scheduling
  .....
- constant folding
  scheduling
  .....

*Iteration of this process to search for the fastest*