Decoupled Access/Execute Metaprogramming for GPU-Accelerated Systems

Anton Lokhmotov\textsuperscript{1} \hspace{1cm} Lee Howes\textsuperscript{1}
Alastair F. Donaldson\textsuperscript{2,3} \hspace{1cm} Paul H.J. Kelly\textsuperscript{1}

Imperial College
London

\textsuperscript{1}

Urbana-Champaign, 28 July 2009
Recent meeting on accelerated computing at Imperial
(35–40 attendees summarised by their affiliation)

- Computing (software optimisation, cognitive robotics, visual information processing, reconfigurable computing)
- Electrical Engineering (reconfigurable computing, design automation)
- Mechanical Engineering (multiscale flow dynamics, vibration technology)
- Earth Science and Engineering (applied modelling & computation)
- Physics (plasma, experimental solid state)
- Chemistry (computational, biological & biophysical)
- Biomedical Engineering
- Chemical Engineering
- Civil Engineering
- Aeronautics
Why accelerator programming is challenging?

Accelerator hardware
- hundreds of functional units
- software-managed memory hierarchies, e.g.
  - host memory (main memory)
  - device global memory (on-board)
  - device local memory (on-chip)

Accelerator software
- low-level, hence unproductive
- architecture-specific, hence nonportable
The fundamental software engineering challenge

How to use accelerator technology but keep

- maintainability,
- composability,
- reusability,
- portability?
Motivating example A: vertical mean image filter

\[ O_{x,y} = \frac{1}{D} \sum_{k=0}^{D-1} I_{x,y+k} \]

- \( I \) is a \( W \times H \) grey-scale input image;
- \( O \) is a \( W \times (H - D) \) grey-scale output image;
- \( D \) is the diameter of the filter \( (D \ll H) \);
- \( 0 \leq x < W, \ 0 \leq y < H - D \).
Memory access of iteration \((x, y)\) \([D = 2]\)

\[
O_{x,y} = \frac{1}{D} \sum_{k=0}^{D-1} I_{x,y+k} = \frac{1}{2} (I_{x,y} + I_{x,y+1})
\]
Scalable algorithm for vertical mean image filter

$$O_{x,y} = \begin{cases} \frac{1}{D} \sum_{k=0}^{D-1} I_{x,y+k}, & \text{for } y = y_0; \\ O_{x,y-1} + \frac{1}{D}(I_{x,y+D-1} - I_{x,y-1}), & \text{for } 1 \leq y - y_0 < T. \end{cases}$$

```
for(int x = 0; x < W; ++x) { // for each column
    for(int y0 = 0; y0 < H-D; y0 += T) { // for each strip of rows
        // first phase: convolution
        float sum = 0.0f;
        for(int k = 0; k < D; ++k)
            sum += I[(y0+k)*W + x];
        O[y0*W + x] = sum / (float)D;

        // second phase: rolling sum
        for(int dy = 1; dy < min(T,H-D-y0); ++dy) {
            int y = y0 + dy;
            sum -= I[(y-1)*W + x];
            sum += I[(y-1+D)*W + x];
            O[y*W + x] = sum / (float)D;
        }
    }
}
```
Complexity and parallelism

\[ O_{x,y} = \begin{cases} \frac{1}{D} \sum_{k=0}^{D-1} I_{x,y+k}, & \text{for } y = y_0; \\ O_{x,y-1} + \frac{1}{D} (I_{x,y+D-1} - I_{x,y-1}), & \text{for } 1 \leq y - y_0 < T. \end{cases} \]

- Writes to \( O \): \( N = W \times (H - D) \)
- Reads from \( I \) and arithmetic operations: \( \Theta(N + ND/T) \)
  - \( \Theta(ND) \) when \( T \ll D \)
  - \( \Theta(N) \) when \( T \gg D \)
- Thread parallelism: \( \lceil N/T \rceil \)
Each WPBX × WPBY box is assigned to a thread block.

work/block = work/thread × threads/block, e.g.

- WPBX = 1 pixel × 128 threads/block = 128 pixels/block
- WPBY = T pixels × 1 thread/block = T pixels/block

Red regions represent inefficiency (early terminated threads)
One-dimensional grid mapping

- alleviating inefficiency by wrapping around the right edge
- wasting a small number of threads to keep SIMD alignment
VMIF on NVIDIA GTX 280, 5120 × 3200 image
2D grid; 1D grid.

Throughput (Mpixel/s)

Output pixels per thread (T)

vmean: W=5120, H=3200, D=40, TPBX=128, TPBY=1

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
VMIF on NVIDIA GTX 280, 5121 × 3200 image
2D grid. Data padded to multiples of 16, 32, 64, and 128 ps.
VMIF on NVIDIA GTX 280, 5121 × 3200 image
Data padded to a multiple of 64 ps. 1D grid wrapped on multiples of 16, 32 and 64 ps.
VMIF on NVIDIA GTX 280, $5121 \times 3200$ image

Data padded to a multiple of 64 ps. 2D grid; 1D grid wrapped on a multiple of 32 ps.

Throughput (Mpixel/s) vs. Output pixels per thread (T) for vmean: $W=5121$ (padded to 5184), $H=3200$, $D=40$, $TPBX=128$, $TPBY=1$.
Motivating example B: horizontal mean image filter

\[ O_{x,y} = \frac{1}{D} \sum_{k=0}^{D-1} I_{x+k,y} \]

- \( I \) is a \( W \times H \) grey-scale input image;
- \( O \) is a \( (W-D) \times H \) grey-scale output image;
- \( D \) is the *diameter* of the filter \( (D \ll W) \);
- \( 0 \leq x < W - D, \ 0 \leq y < H \).
Bad news and good ideas

Bad news

- Threads in a warp access consecutive *rows*: non-coalesced memory access results in reduced bandwidth

Good ideas

- use shared memory to repair inefficient memory access
- use optimised vertical mean filter implementation:
  - \( \text{hmean} := \text{transpose} \mid \text{vmean} \mid \text{transpose} \)
hmean := transpose | vmean | transpose

---

Forward transpose: $5120 \times 3200 \rightarrow 3200 \times 5120$

- Output pitch 3200: 60.2 GiB/s
- Output pitch 3264 = 3200+64: 81.1 GiB/s

Backward transpose: $3200 \times 5120 \rightarrow 5120 \times 3200$

- Output pitch 5120: 19.1 GiB/s
- Output pitch 5184 = 5120+64: 63.3 GiB/s
- Output pitch 5248 = 5120+128: 50.9 GiB/s
HMIF on NVIDIA GTX 280, 5120 × 3200 image

hmean: W=5120, H=3200, D=40

Vanilla
Transpose
Shared memory, 32 threads
Shared memory, 64 threads

Throughput (Mpixel/s) vs. Output pixels per thread (T)
Towards metaprogramming

- Separate algorithm representation from mapping and tuning
- Generate efficient device-specific code, in particular:
  - synthesise data movement for software-managed memory hierarchy
  - handle device-specific alignment, padding, etc.
Decoupled Access/Execute (Æcute) model

Decoupled Access/Execute metaprogramming

- kernel code written for uniform memory
- execute metadata describe execution constraints
- access metadata describe memory access pattern
- part of the kernel’s interface specification

Goals

- robust translation into efficient low-level code
- ample opportunities for optimisation
- convenience and flexibility
Execute metadata

Execute metadata for a kernel is a tuple $E = (I, R, P)$, where:

- $I \subseteq \mathbb{Z}^n$ is a finite, $n$-dimensional iteration space, for some $n > 0$;
- $R \subseteq I \times I$, is a precedence relation such that $(i_1, i_2) \in R$ iff iteration $i_1$ must be executed before iteration $i_2$.
- $P$ is a partition of $I$ into a set of non-empty, disjoint iteration subspaces: $P = \{ I_k : I = \bigcup I_k; I_i \cap I_j = \emptyset, i \neq j \}$
Access metadata

Access metadata for a kernel is a tuple $A = (M_r, M_w)$, where:

- $M_r : I \rightarrow \mathcal{P}(M)$ specifies the set of memory locations $M_r(i)$ that may be *read* on iteration $i \in I$;
- $M_w : I \rightarrow \mathcal{P}(M)$ specifies the set of memory locations $M_w(i)$ that may be *written* on iteration $i \in I$. 

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly

Decoupled Access/Execute Metaprogramming
Example: 2D convolution

\[ O_{y,x} = \sum_{u=-K}^{K} \sum_{v=-K}^{K} C_{u,v} \cdot I_{y+u,x+v} \]

- \( I \): input image
- \( O \): output image
- \( C \): coefficients
- \( W \): image width
- \( H \): image height
- \( K \): neighbourhood radius
- \( K \leq y < H - K; K \leq x < W - K \)
Memory access of a single \((y, x)\) iteration \((K = 1)\)

\[
O_{y,x} = \sum_{u=-K}^{K} \sum_{v=-K}^{K} C_{u,v} \cdot I_{y+u, x+v}
\]
Æcute specification ($h \times w$ rectangular tiling)

\[
O_{y,x} = \sum_{u=-K}^{K} \sum_{v=-K}^{K} C_{u,v} \cdot I_{y+u,x+v}
\]

Execute metadata ($I, R, P$):

- $I = \{(y, x) : K \leq y < H - K, K \leq x < W - K\}$
- $R = \emptyset$
- $P = \{\{(y, x) \in I : h(j - 1) \leq y - K < hj, w(i - 1) \leq x - K < wi\} : 1 \leq j < (H - 2K)/h, 1 \leq i < (W - 2K)/w\}$

Access metadata ($M_r, M_w$):

- $M_r = \{I_{y+u,x+v}, C_{u,v} : (y, x) \in I, -K \leq u, v \leq K\}$
- $M_w = \{O_{y,x} : (y, x) \in I\}$
\[ i, j, O_{i,j}: 0 \leq i < H, 0 \leq j < W; C_{i,j}: -K \leq i, j \leq K \]

**C++**

```cpp
rgb I[W][H];
rgb O[W][H];
rgb C[2*K+1][2*K+1];
```

**Æcute (data wrappers)**

```cpp
Array2D<rgb> arrayI(&I[0][0], W, H);
Array2D<rgb> arrayO(&O[0][0], W, H);
Array2D<rgb> arrayC(&C[0][0], 2*K+1, 2*K+1);
```
Iteration space: $K \leq y < H - K, K \leq x < W - K$

C++

```cpp
for(y = K; y < H-K; ++y)
    for(x = K; x < W-K; ++x)
        // Kernel code for each (y, x)
```

Æcute (execute metadata)

```cpp
IterationSpace1D y(K,H-K);
IterationSpace1D x(K,W-K);
IterationSpace2D iterYX(y,x);
```
Access regions: implicit in C++, explicit in Æcute

C++

// Kernel code for each (y,x)
rgb sum(0.0f, 0.0f, 0.0f);
for (u = -K; u <= K; ++u)
    for (v = -K; v <= K; ++v)
        sum += C[K+u][K+v] * I[y+u][x+v]; // read from C and I
O[y][x] = sum; // write to O

Æcute (access metadata)

// Access descriptors
Neighbourhood2D_R accessI(iterYX, arrayI, K);
Point2D_W accessO(iterYX, arrayO);
All_R accessC(iterYX, arrayC);
Kernel code

C++

// Kernel code for each (y,x)
int u, v;
rgb sum(0.0f, 0.0f, 0.0f);
for (u = -K; u <= K; ++u)
    for (v = -K; v <= K; ++v)
        sum += C[K+u][K+v] * I[y+u][x+v];
O[y][x] = sum;

Æcute (kernel method)

void kernel(const IterationSpace2D::iterator &it)
{
    int u, v;
    rgb sum(0.0f, 0.0f, 0.0f);
    for (u = -K; u <= K; ++u)
        for (v = -K; v <= K; ++v)
            sum += accessC(u, v) * accessI(it, u, v);
    accessO(it) = sum;
}
Bringing all together

// Data wrappers
Array2D<rgb> arrayI(&I, W, H);
Array2D<rgb> arrayO(&O, W, H);
Array2D<rgb> arrayC(&C, 2*K+1, 2*K+1);

// Execute metadata
IterationSpace1D y(K,H-K);
IterationSpace1D x(K,W-K);
IterationSpace2D iterYX(y,x);

// Access metadata
Neighbourhood2D_R accessI(iterYX, arrayI, K);
Point2D_W accessO(iterYX, arrayO);
All_R accessC(iterYX, arrayC);

// Filter initialisation and execution
ConvolutionFilter2D conv(iterYX, accessI, accessO, accessC);
iterYX.tile(h, w);
conv.run();
Æcute metadata benefits

- data movement synthesis and optimisation (e.g. software pipelining and exploiting data reuse)
- machine-independent abstraction, machine-dependent tuning (via partitioning)
- potential for inter-kernels optimisations (e.g. loop fusion and array contraction)
Memory access of an iteration subspace

Let $I_k \subset I$ be an iteration subspace

- $M_r(I_k) = \bigcup_{i \in I_k} M_r(i)$
- $M_w(I_k) = \bigcup_{i \in I_k} M_w(i)$
Let $I_k$ and $I_n$ be two iteration subspaces

- $M_r(I_k) \cap M_r(I_n)$ determines reuse
- $M_w(I_k) \cap M_r(I_n)$ determines true dependence
- $M_r(I_k) \cap M_w(I_n)$ determines anti dependence
- $M_w(I_k) \cap M_w(I_n)$ determines output dependence
Inter-kernel optimisation (fusion and contraction)

Region of $I$

Region of $T$

Region of $O$

Kernel A

Kernel B

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Inter-kernel optimisation (fusion and contraction)

- Improved locality, reduced communication
- Tricky but within the reach of polyhedral code generation
- Using metadata bypasses fragile inter-kernel analysis
Current & future work

- Generating and optimising code for multiple targets (OpenCL)
- Integrating Æcute metadata into existing languages
  - Sieve C++ (EPSRC CASE Award Imperial/Codeplay)
  - OpenCL (AMD)
  - OpenMP?
- Handling both regular and irregular computations
- Extracting metadata both statically and dynamically
- Using metadata for cross optimisation of parallel components
- Scaling to large applications
References


Questions?
Prototype framework for the Cell BE architecture

Sony/Toshiba/IBM Cell Broadband Engine

- 1 PowerPC Processing Element (PPE)
- 8 Synergistic Processing Elements (SPEs)
- Main memory, 256 KiB local memory per SPE
- DMA to copy data between main and local memories

Benchmark variants (kernel code is basically the same)

- æcute
- Hand-written C
- Software cache (IBM Cell SDK)

Experimental setup

- 3.2 GHz Cell (Sony PlayStation 3, 6 SPEs available)
- IBM Cell SDK 2.1, Fedora Linux 7
Prototype framework for the Cell BE architecture
PPE and SPE runtimes

Data in main memory
Buffers
Input access descriptors

Output access descriptors
Buffers
Data in main memory
Prototype framework for the Cell BE architecture

PPE takes a block of the iteration space (blocking is configurable)
Prototype framework for the Cell BE architecture

PPE assigns the block of iterations to a ready SPE by sending a message.
Prototype framework for the Cell BE architecture
SPE copies the block’s working set into buffers associated with access descriptors

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Prototype framework for the Cell BE architecture

Whilst SPE processes the block, it receives another block for execution.
Prototype framework for the Cell BE architecture

SPE copies the new block’s working set into another set of buffers

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Prototype framework for the Cell BE architecture
SPE copies out the first block’s results and clears its buffers

Data in main memory

Buffers

Input access descriptors

PPE
Runtime

Iteration space

SPE
Runtime

Output access descriptors

Buffers

Data in main memory
Matrix-vector multiply: \( \vec{y} = A \cdot \vec{x} \)

- Bandwidth limited
- Tiling increases locality: \( \vec{x} \) is reused
- Longer strips (e.g. 1024) are more efficient
- \( \text{Æcute} \) is 2.3–3.6× slower, software cache is 5–10× slower

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Closest-to-mean filter on square $D \times D$ images

No tile size was universally best

- 40 $\times$ 5 is best for $D=256$
- 20 $\times$ 20 is best for $D=1024$
- 5 $\times$ 40 is near best for both $D=256$ and 1024

Æcute is 12–40% slower, software cache is 2.2–9.6 $\times$ slower
Bit-reversed copy: \( y[\sigma_n(i)] = x[i] \), for \( 0 \leq i < N = 2^n \)

\( (\sigma_n(i)) \) reverses bits of \( n \)-bit index, e.g. \( \sigma_4(0111_2) = 1110_2 \)

- Tiled algorithm (Carter & Gatlin)
- Vector kernel (Lokhmotov & Mycroft)
- Non-affine access specification
- For large problem sizes \( (n = 22–24) \), Æcute is \( 1.6 \times \) slower, software cache is \( 20 \times \) slower
Matrix-vector multiply

Execute metadata \((I, R, P)\):

- \(I = \{(i, j) : 0 \leq i < H, 0 \leq j < W\}\)
- \(R = \{((i, j), (i, k)) : 0 \leq i < H, 0 \leq j < k < W\}\)
- \(P = \{\{(i, j) \in I : h(k - 1) \leq i < hk, w(l - 1) \leq j < wl\} : 1 \leq k < H/h, 1 \leq l < W/w\}\)

Access metadata \((M_r, M_w)\):

- \(M_r(i, j) = \{A[i][j], x[j]\}\)
- \(M_w(i, j) = \{y[i]\}\)
Closest-to-mean filter

Execute metadata \((I, R, P)\):

- \(I = \{(x, y) : K \leq x < W - K, K \leq y < H - K\}\)
- \(R = \emptyset\)
- \(P = \{((x, y)) \in I : w(i - 1) \leq x - K < w_i, h(j - 1) \leq y - K < h_j : 1 \leq i < (W - 2K)/w, 1 \leq j < (H - 2K)/h\}\)

Access metadata \((M_r, M_w)\):

- \(M_r = \{I[x + u][y + v] : (x, y) \in I, -K \leq u, v \leq K\}\)
- \(M_w = \{O[x][y] : (x, y) \in I\}\)
Bit-reversed copy

Execute metadata \((I, R, P)\):

- \(I = \{t : 0 \leq t < N / B^2\}\)
- \(R = \emptyset\)
- \(P = \{\{t\} : t \in I\}\)

Access metadata \((M_r, M_w)\):

- \(M_r(t) = \{x[u.t.v] : t \in I, 0 \leq u, v < B\}\)
- \(M_w(t) = \{y[u.\sigma_n(t).v] : t \in I, 0 \leq u, v < B\}\)
Parallel scopes
Joint work with A. Mycroft (Cambridge), A. Donaldson, A. Richards (Codeplay)

- Aliasing complicates dependence analysis
- The compiler must produce reliable, if inefficient, code
- The programmer needs to tell more to the compiler
Delayed side-effects in Codeplay’s Sieve C++

In Codeplay’s C++ extension, the programmer can place a code fragment in a *parallel scope*, thereby instructing the compiler to

- *delay* writes to memory locations defined outside of the scope, and
- apply them *in order* on exit from the scope.

---

**Sieve C++**

```cpp
defloat *pa, *pb; ...
sieve { // sieve block
    for(int i = 1; i <= n; ++i)
    {
        pb[i] = pa[i] + 42;
    }
} // writes to pb[1:n] happen here
```

**Fortran 90**

```fortran
pb[1:n] = pa[1:n] + 42;
```
Example: N-body simulation
Force acting on particle $i$ according to the law of gravity

$$\vec{F}_i = - \sum_{j \neq i} G \frac{M_i M_j}{|\vec{r}_{ij}|^3} \vec{r}_{ij}$$
Example: N-body simulation

C++ code

```cpp
int Size;
float3 rNormalised(float3 * Pos, int i, int j);
...
void computeForces(float3 * Force, float3 * Pos, float * Mass) {
    for(int i = 0; i < Size; ++i) {
        float3 Potential = { 0.0f, 0.0f, 0.0f };  
        for(int j = 0; j < Size; ++j) {
            float3 r = rNormalised(Pos, i, j);
            Potential -= r * Mass[j];
        }
        Force[i] = Potential * Mass[i];
    }
}
```
Example: N-body simulation
Sieve C++ code

```cpp
int Size;
sieve float3 rNormalised(outer float3 * Pos, int i, int j);
...
void computeForces(float3 * Force, float3 * Pos, float * Mass) {
    sieve
    {
        for(int i = 0; i < Size; ++i) {
            float3 Potential = { 0.0f, 0.0f, 0.0f };  
            for(int j = 0; j < Size; ++j) {
                float3 r = rNormalised(Pos, i, j);
                Potential -= r * Mass[j];
            }
            // Delayed write
            Force[i] = Potential * Mass[i];
        }
        // Side-effects to Force[] committed here
    }
}
```

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Sieve/OpenMP C++ (Codeplay/Microsoft) vs. C++
Dual quad-core AMD Opteron (2GHz Barcelona), 4GiB RAM, Windows Server 2003

Benchmark details

A. Lokhmotov, L. Howes, A.F. Donaldson, P.H.J. Kelly
Decoupled Access/Execute Metaprogramming
Sieve/OpenMP C++ (Codeplay/IBM XL Alpha) vs. C++
Sony PlayStation 3 (3.2GHz Cell BE, 6 SPEs), IBM SDK 3, Fedora Core 7

Number of active SPE cores/PPE+SPE threads used by Sieve/OpenMP shown above each bar

Benchmark details

Lessons learnt

The concept of delayed side-effects is useful
- Eliminates the need for fragile whole-program analysis
- Enables safe speculative parallelisation
- Provides deterministic behaviour

Efficient implementation is hard
- Data movement is the culprit for heterogeneous architectures
- Mainstream languages mix memory access and compute operations
- Efficient programs decouple memory access and computation (and then overlap them asynchronously)

Conclusion: we need a programming model that promotes this decoupling (hence Æcute)!
Sieve/OpenMP C++ benchmarks

- **GRAVITY**: $N$-body molecular dynamics simulation of 8192 particles
- **NOISE RGB**: Noise reduction filter applied to $512 \times 512$ colour image
- **NOISE GREY**: Noise reduction filter applied to $512 \times 512$ greyscale image
- **CRC**: Cyclic redundancy check on random 8M (1M=$2^{20}$) word message
- **MAND**: Calculates $1024 \times 1024$ fragment of the Mandelbrot set
- **FFT3D**: Fast Fourier transform of complex $128^3$ data set
Bit-reversal on NVIDIA GTX 280

\[ y[\sigma_n(i)] = x[i], \text{ for } 0 \leq i < N = 2^n \]