Kurs:

Procesory graficzne w obliczeniach równoległych (CUDA)

Wykład 5:
obliczenia float, atomiki,
potencjał elektrostatyczny,
narzędzia,

Obliczenia: float

- Operacje zmiennoprzecinkowe na GPU G80:

  standard IEEE-754 ale z odchyleniami

  inna dokładność niż na CPU

  NaN jest, zamiast Inf.,-Inf wartość max/min

  pierwiastek kwadrat. i dzielenie: software

- Obliczenia na double wolniejsze,

  zaimplementowane od kart serii G200

Obliczenia: float

- Operacje int/float:
  add,shift,mul,min,max,mad - 4 cykle/warp

  - mnożenie int 32-bitowe wymaga więcej cykli

  - wersje 24-bitowa (4cykle): __mul24(), __umul24()

- Dzielenie (także int) i operacje modulo drogie

  dla `n` będącego potegą 2:

  - dzielenie przez `n` kompilator zamieni na shift'y

  - ale zamiast: `x%n` szybciej `x&(n-1)`

Obliczenia: float

- Odwrotność, odwrotność pierwiastka kwadratowego, sin, cos, log, exp (wersje szybkie: __sin(),..) wykonywane są w 16 cyklach/warp
- Inne operacje są dla G80 kombinacjami:

  y/x = rcp(x) * y - 20 cykli

  sqrt(x) = rcp(rsqrt(x)) - 32 cykle

Obliczenia: float

- Funkcje typu `__sin()` są sprzętowe
- Flaga kompilatora: -use_fast_math

  `sin()` → `__sin()`
- Funkcje typu `sin()` są dokładniejsze ale realizowane programowo i wolniejsze
- Double (od G200) wolniejsze wiec aby uniknąć bezpieczniej napisać explicite:

```
y = sinf(x);        y = x * 0.001f

y = rsqrtf(x);
```

Odchylenia od IEEE-754

- Mnożenie i dodawanie zgodne, ale jeśli zastąpione przez FMAD to już nie (wynik pośredni obcięty).
- Dzielenie dla G80 nie zgodne ze standardem (błąd większy), dla Fermi : G400 zgodne
- Dużo funkcji z błędem niezgodnym ze stand.

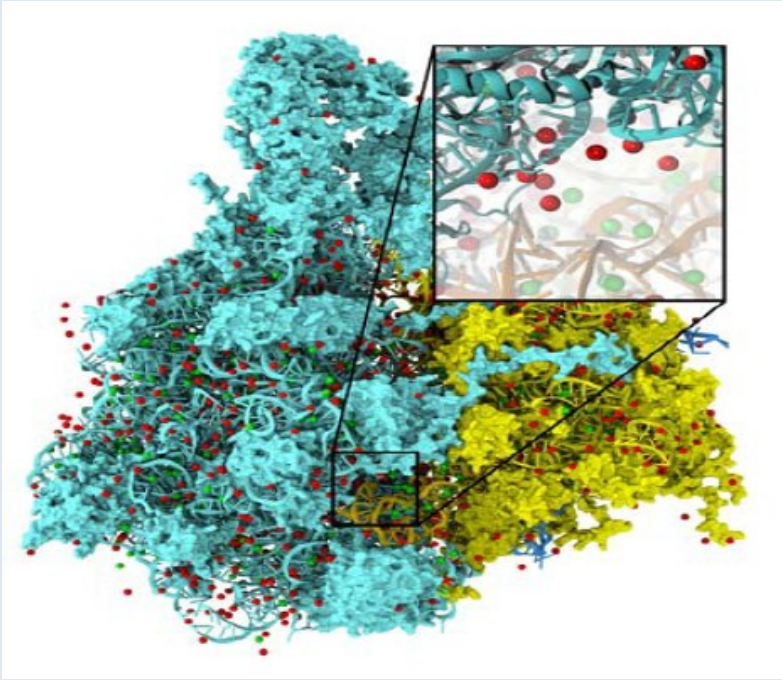
  → dokumentacja
- Nie wszystkie tryby zaokrąglania, brak Inf.

Atomiki (od ~ G200)

- Operacje atomowe gwarantują wykonanie sekwencji `read-modify-write` bez ingerencji innych wątków
- Operacje wykonywane na 32 lub 64 bitowym słowie w pamięci global lub shared, głównie na int'ach
- atomicAdd: `int atomicAdd(int *adres, int wartosc);`
- atomicSub(), atomicExch(), atomicMin(),...
- atomicAnd(), …
- Koszt większy i są od generacji G200 (C.C. ver. 1.2)

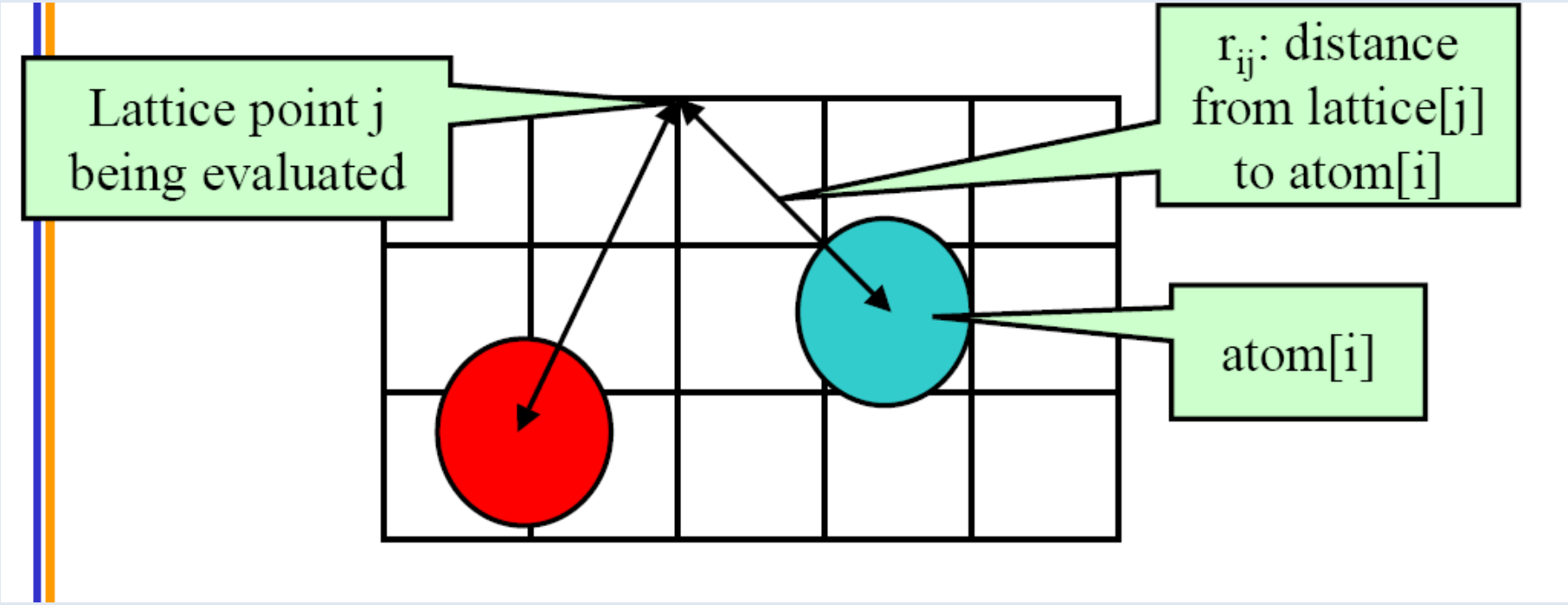
Przykład (Wu/Kirk): obliczenia potencjału elektrostatycznego

- Molecular Dynamics Simulation
- VMD software – Visual Molecular Dynamics
- Umieszczanie jonów/dokowanie



Przykład (Wu/Kirk): obliczenia potencjału elektrostatycznego

- Dla regularnej kraty punktów obliczamy potencjał: sumę po wszystkich atomach,
  DSC (Direct Coulomb Summation), koszt:...
- Wpływ atomu to ładunek/odległość



DCS dla CPU (Kirk/Wu):

```
void cenergy(float *energygrid, dim3 grid, float gridspacing, float z, const float *atoms,
             int numatoms) {
  int i,j,n;
  int atomarrdim = numatoms * 4;
  for (j=0; j<grid.y; j++) {
    float y = gridspacing * (float) j;
    for (i=0; i<grid.x; i++) {
      float x = gridspacing * (float) i;
      float energy = 0.0f;
      for (n=0; n<atomarrdim; n+=4) {     // calculate potential contribution of each atom
        float dx = x - atoms[n    ];
        float dy = y - atoms[n+1];
        float dz = z - atoms[n+2];
        energy += atoms[n+3] / sqrtf(dx*dx + dy*dy + dz*dz);
      }
      energygrid[grid.x*grid.y*k + grid.x*j + i] = energy;
    }
  }
}
```

DCS dla GPU (Kirk/Wu) :

- Wątek może obliczać:

  wpływ jednego atomu na kratę:

  wymagane są atomiki i różne wątki pisza w jedno miejsce

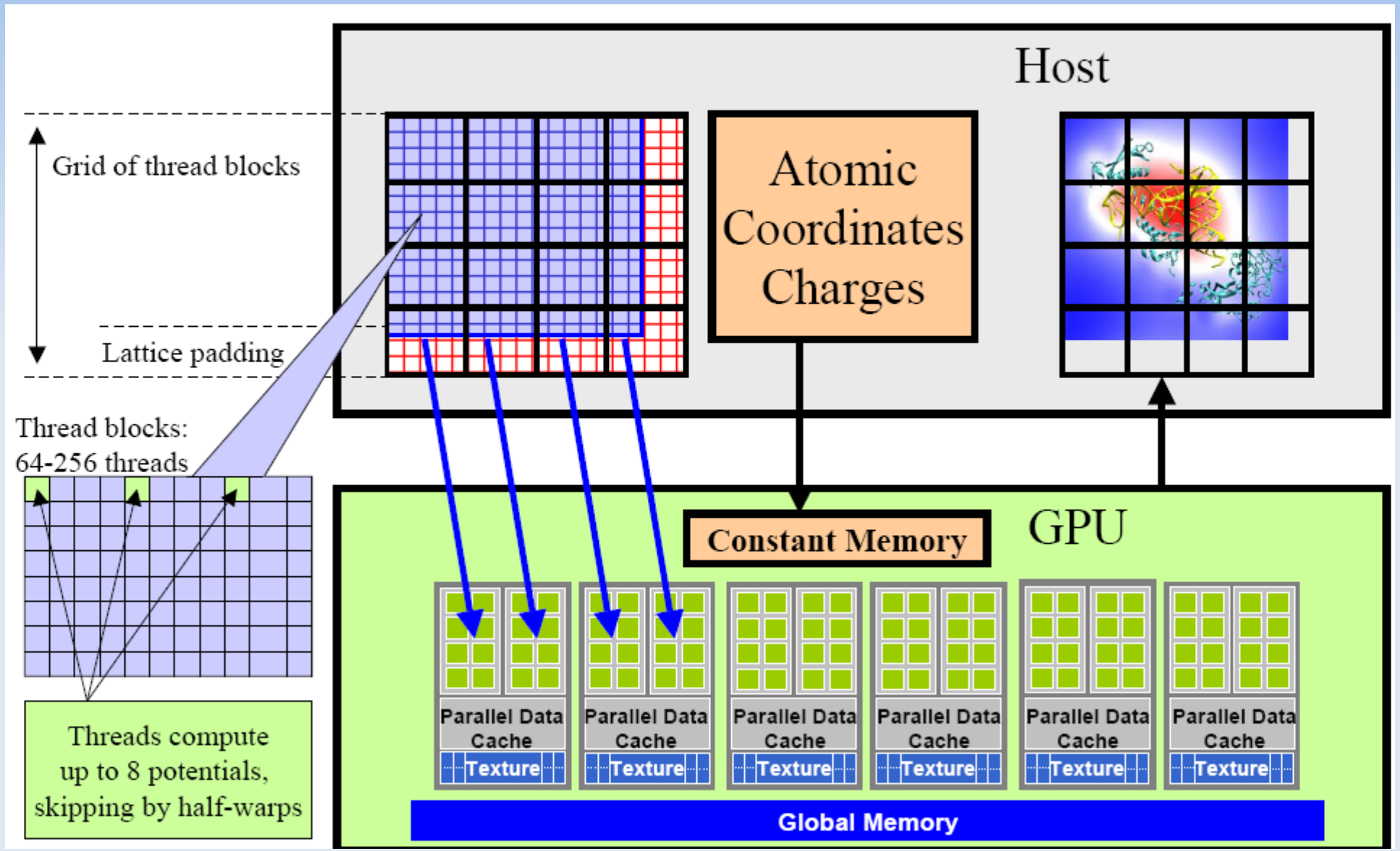
  → potencjał w jednym punkcie kraty:

  iterując po atomach (wspólne czytanie lepsze niż pisanie)

- Obliczamy wpływ wszystkich atomów:

  są też algorytmy które ograniczają odległość

DCS dla GPU (Kirk/Wu) :



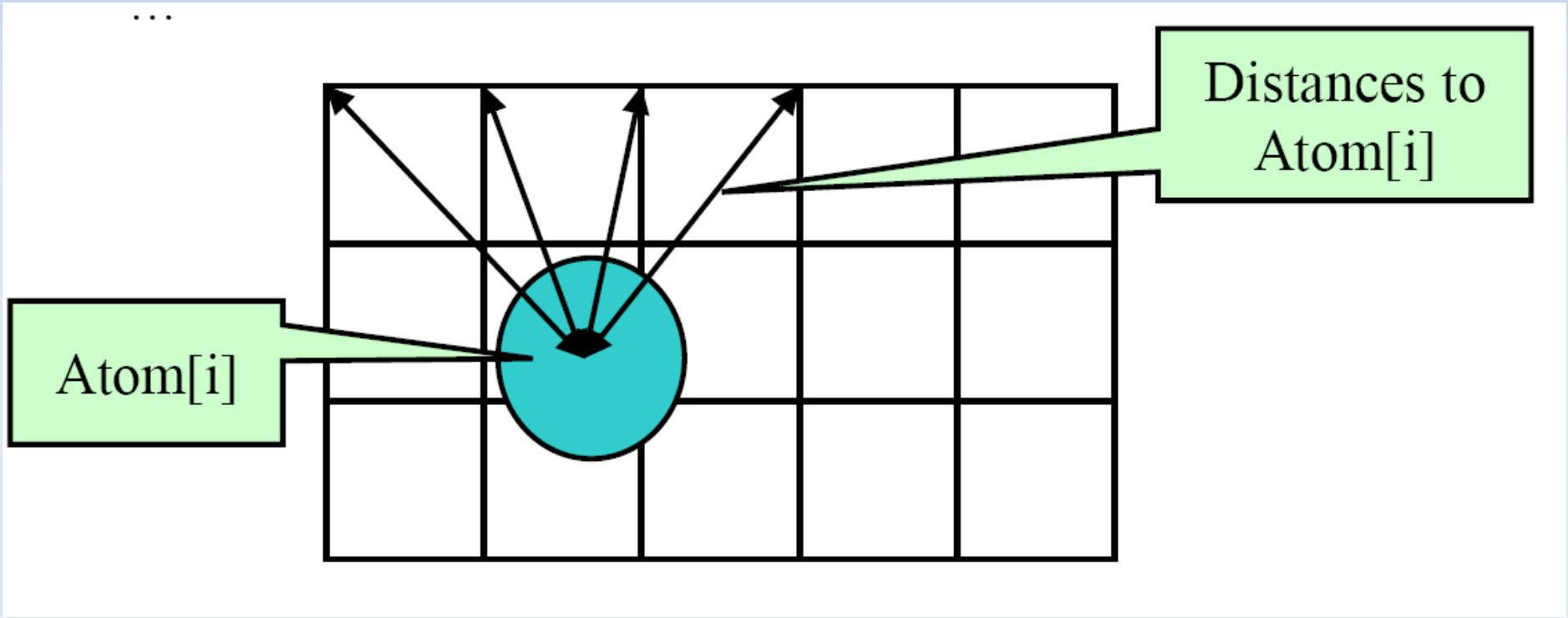
DCS dla GPU ver.1 (Kirk/Wu) :

```
...
 float curenergy = energygrid[outaddr];
 float coorx = gridspacing * xindex;
 float coory = gridspacing * yindex;
 int atomid;
 float energyval=0.0f;
  for (atomid=0; atomid<numatoms; atomid++) {
   float dx = coorx - atominfo[atomid].x;
   float dy = coory - atominfo[atomid].y;
   energyval += atominfo[atomid].w *
                        rsqrtf(dx*dx + dy*dy + atominfo[atomid].z);
 }
 energygrid[outaddr] = curenergy + energyval;
```

Start global memory reads early. Kernel hides some of its own latency.

Only dependency on global memory read is at the end of the kernel…

Wielokrotne użycie danych

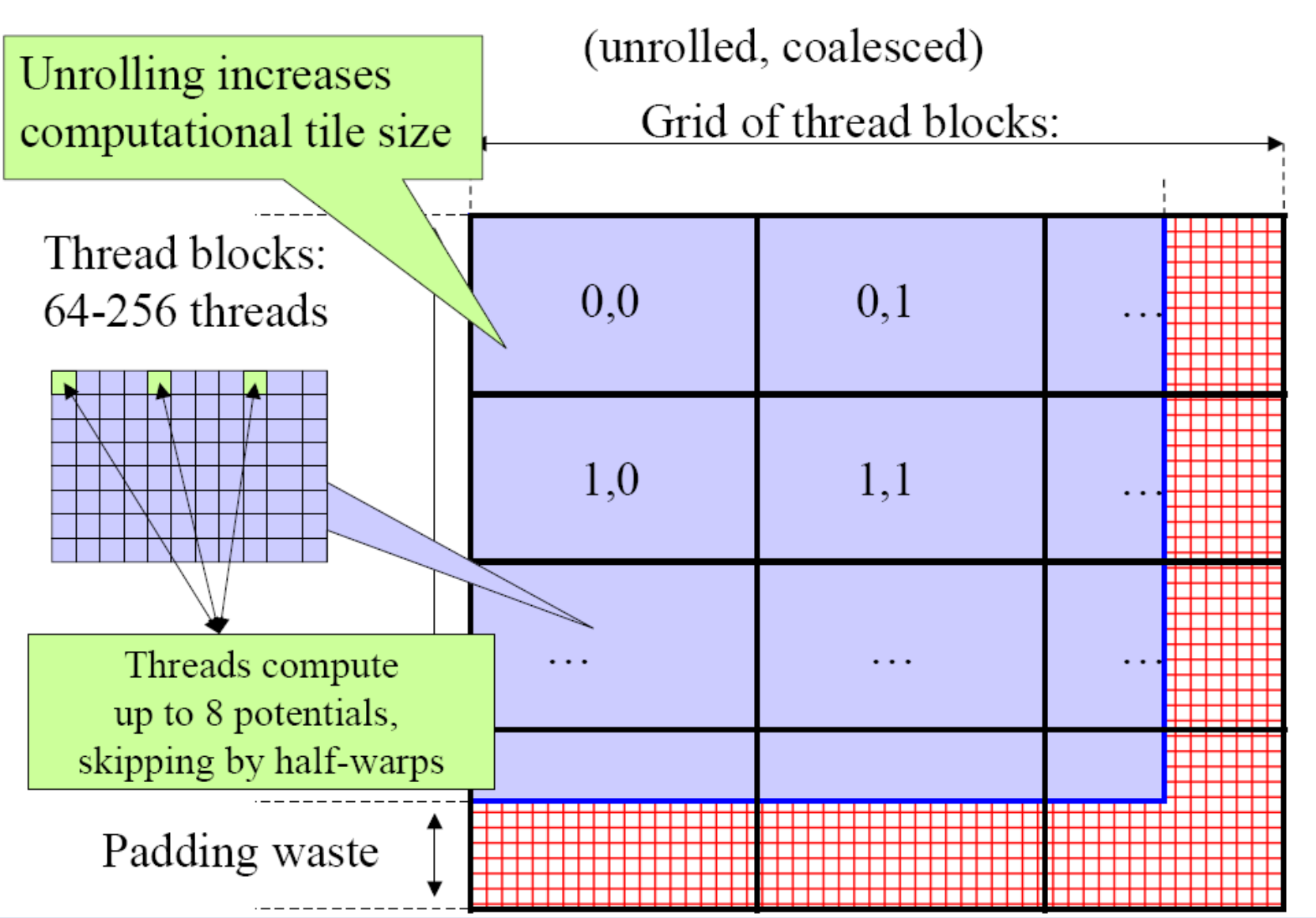


DCS dla GPU ver.2 (Kirk/Wu) :

```
…for (atomid=0; atomid<numatoms; atomid++) {
    float dy = coory - atominfo[atomid].y;
    float dysqpdzsq = (dy * dy) + atominfo[atomid].z;
    float x = atominfo[atomid].x;
    float dx1 = coorx1 - x;
    float dx2 = coorx2 - x;
    float dx3 = coorx3 - x;
    float dx4 = coorx4 - x;
    float charge = atominfo[atomid].w;
    energyvalx1 += charge * rsqrtf(dx1*dx1 + dysqpdzsq);
    energyvalx2 += charge * rsqrtf(dx2*dx2 + dysqpdzsq);
    energyvalx3 += charge * rsqrtf(dx3*dx3 + dysqpdzsq);
    energyvalx4 += charge * rsqrtf(dx4*dx4 + dysqpdzsq);
  }
```

Compared to non-unrolled kernel: memory loads are decreased by 4x, and FLOPS per evaluation are reduced, but register use is increased…

Pamięć: coalescing (Kirk/Wu)



DCS dla GPU ver.3 (Kirk/Wu) :

```
…float coory = gridspacing * yindex;
   float coorx = gridspacing * xindex;
   float gridspacing_coalesce = gridspacing * BLOCKSIZEX;
   int atomid;
   for (atomid=0; atomid<numatoms; atomid++) {
     float dy = coory - atominfo[atomid].y;
     float dyz2 = (dy * dy) + atominfo[atomid].z;
     float dx1 = coorx - atominfo[atomid].x;
[…]
     float dx8 = dx7 + gridspacing_coalesce;
     energyvalx1 += atominfo[atomid].w * rsqrtf(dx1*dx1 + dyz2);
[…]
     energyvalx8 += atominfo[atomid].w * rsqrtf(dx8*dx8 + dyz2);
   }
  energygrid[outaddr                    ] += energyvalx1;
[...]
  energygrid[outaddr+7*BLOCKSIZEX] += energyvalx7;
```

Points spaced for memory coalescing

Reuse partial distance components dy^2 + dz^2

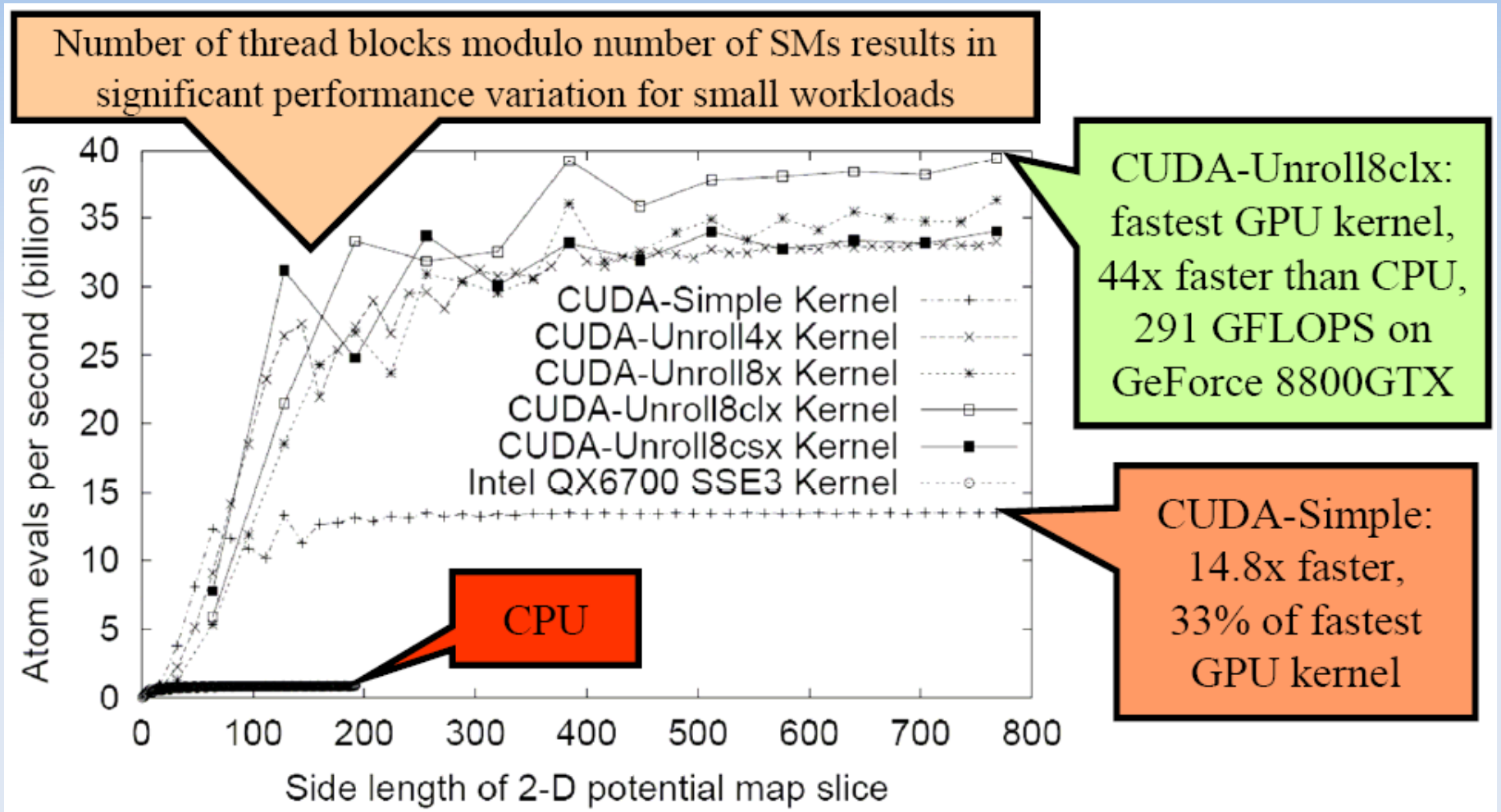
Global memory ops occur only at the end of the kernel, decreases register use

DCS dla GPU podsumowanie :

- Jądro oblicza plasterek x-y kraty dla danego z
- Każdy wątek iteruje po wszystkich atomach
- Dane o atomach umieszczamy w cachowanej pamięci stałych: tylko 64KB:

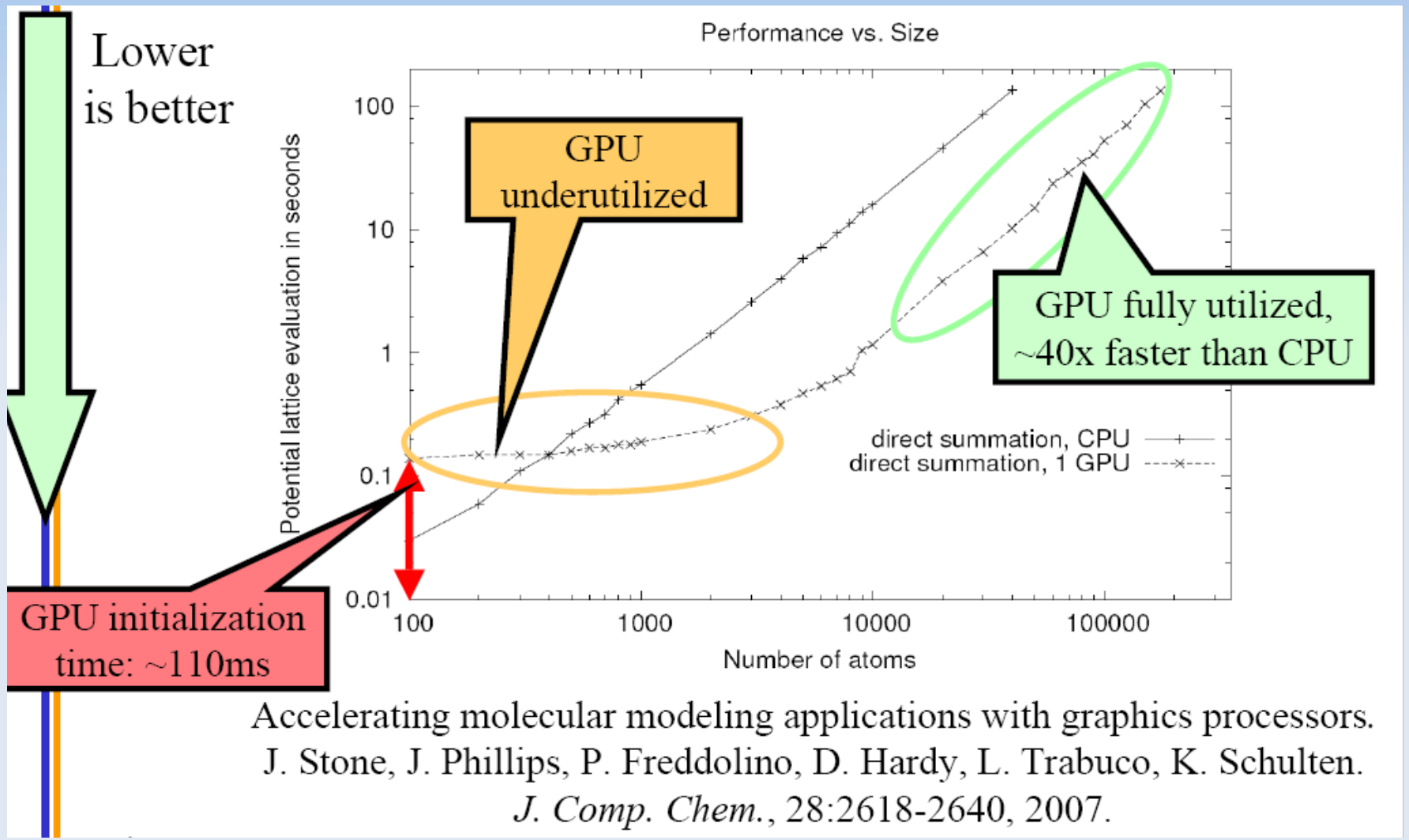
  jak atomów więcej to uruchamiamy kilka razy
- Każdy wątek po wczytaniu każdego atomu liczy kilka elementów kraty
- Elementy kraty w wątku nie kolejne ale z przeskokiem aby zapewnić coalescence
- Wymiary tablic uzupełniamy dla alignement

Wyniki dla DCS :



GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.

Wyniki dla DCS: GPU vs. CPU



Accelerating molecular modeling applications with graphics processors.
J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten.
J. Comp. Chem., 28:2618-2640, 2007.

Narzędzia

- CUDA-gdb

  wersja debuggera gdb ze wsparciem CUDA
- CUDA-Memcheck (dopiero od CUDA 3.0)

  dostępny osobno i zintegrowany z cuda-gdb
- CUDA Visual Profiler
- Nexus: debugger dla microsoft windows, integracja z Visual Studio (chyba już działa?)

Biblioteki

- CUBLAS:

  BLAS- Basic Linear Algebra Subprograms
- CUFFT:

  FFT - Fast Fourier Transform
- MAGMA: Matrix Algebra on GPU and Multicore Architectures (oparta na LAPACK)
- CULA: LAPACK
- PyCUDA: CUDA w pythonie
- Thrust: wersja STL dla CUDA: wektory, iteratory, algorytmy