Praktische CUDA-Programmierung

Johannes Waldmann

HTWK Leipzig

Bereichsseminar 14. 12. 2011

Beispiel

- Aufgabe: gesucht ist
  - Konfiguration von P Punkten im Einheitsquadrat,
  - deren minimaler gegenseitiger Abstand maximal ist.

  (vergleiche `http://www2.stetson.edu/~efriedma/cirinsqu/`)
- Lösungsplan: zufällige lokale Suche
- Teilaufgabe dabei: bewerte eine Liste von K Konfigurationen
- Parallelisierungsmöglichkeiten:
  - Bewertung für alle Konfigurationen gleichzeitig
  - Abstandsberechnung für alle Punktepaare gleichzeitig
- das ist *massiv* parallel ($K = 128, P = 16$) ...
  solche Prozessoren gibt es *preiswert*
  CUDA ist eine C-Erweiterung zur Programmierung dafür

Sequentielle und parallele Lösung (Ansatz)

```
for (int k = 0; k < K; k++) {
  float mindist = +infinity;
  for (int p = 0; p < P; p++) {
    for (int q = p + 1; q < P; q++ ) {
      float s = abstand (c[k][p], c[k][q]);
      mindist = MIN (mindist, s); } }
  c[k].fitness = mindist;  }

dim3 blocks (K, 1) ; dim3 threads (P, P);
kernel <<<blocks,threads>>> (c);
__global__ kernel (config *c) {
  int k = blockIdx.x;
  int p = threadIdx.x; int q = threadIdx.y;
  if (p < q) { float s = abstand (c[k][p], c[k][q])
  // Berechnug des MIN?
}
```

Threadkommunikation

Threads in einem Block haben gemeinsamen *shared memory*

```
__global__ kernel (config *c) {
  __shared__ float dist [P*P];

  if (p < q) dist[p*P+q] = abstand(c[k][p], c[k][q]
  else       dist[p*P+q] = +infinity;
```

Synchronisationspunkt für *alle* (!) Threads in einem Block:

```
  __syncthreads();
```

erst danach kann man Ergebnisse aufsammeln

Binäre Reduktion

sequentiell (linear)

```
float accu = +infty;
for ( ... ) { accu = MIN (accu, dist[ ..]); }
```

parallel (logarithmisch)

```
int tid = P * p + q; int gap = threadsPerBlock / 2;
while ( gap > 0 ) {
  if (dist[tid] > dist[tid + gap])
          dist[tid] = dist[tid + gap];
  __syncthreads ();
  gap /= 2;
}
```

Resultat verarbeiten

```
if (0 == tid) c[k] = dist[0];
```

Datentransport zwischen Host und Decive

```
config * dev_c ;
cudaMalloc ( (void**) &dev_c, K*sizeof(config) );

cudaMemcpy ( dev_c, c, K*sizeof(config),
   cudaMemcpyHostToDevice );
...
cudaMemcpy ( c, dev_c, K*sizeof(config),
   cudaMemcpyDeviceToHost );
cudaFree (dev_c);
```

. . . das ist teuer ($\Rightarrow$ möglichst wenige Daten transportieren)

Zusammenfassung

CUDA-Programmierung ist

- programmiersprachlich einfach
- algorithmisch anspruchsvoll
  (imperative parallele Programmierung)

jeweils projektspezifisch sind festzulegen

- was läuft auf Device, was auf Host?
- Abmessung der Threadblöcke auf dem Device?
- Host realisiert Software-API (Steuerprogramm in Hochsprache)

Alternativen:

- OpenCL
- accelerate/CUDA-backend `http://hackage.haskell.org/package/accelerate`

Literatur

- David B Kirk, Wen-mei W Hwu, *Programming Massively Parallel Processor*, Morgan Kaufmann, 2010
- Jason Sanders, Edward Kandrot: *CUDA by Example*, Addison-Wesley, 2011
  `http://developer.nvidia.com/cuda-example-introduction-general-purpose-gpu-p`