SkePU: A Multi-Backend Skeleton Programming Library for Multi-GPU Systems

Johan Enmyren  Christoph W. Kessler

{x10johen, chrke}@ida.liu.se

Linköping University, Sweden
Outline

- Background
- SkePU
- Evaluation
- Conclusions
- Future Work
Background

- Trend towards multi- and many-core systems
- GPU Computing
- Different parallel programming models for different architectures → portability problem
- Skeleton programming
- SkePU, skeletons for GPU/CPU
SkePU

- C++ template library
- Multiple back ends
- Container
- User functions
- Skeletons
- Multi-GPU
- Various helpers
Container

- Vector-type (modeled after std::vector)
- Handles memory transfers between host and device
- Lazy memory copying

skepu::Vector<double> v0(1000,2);
User Functions

- Macro-language
- Behind the scenes, a C++ struct

```cpp
BINARYFUNC(plus, double, a, b,
    return a+b;
)
Skeletons

- Objects, overloading operator()
- Map
- Reduce
- MapReduce
- MapOverlap
- MapArray
- (Scan)
Map

- Each element in the result vector is a function of the corresponding element in one or more input vectors.

<table>
<thead>
<tr>
<th>Input a</th>
<th>a1</th>
<th>a2</th>
<th>...</th>
<th>an</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input b</td>
<td>b1</td>
<td>b2</td>
<td>...</td>
<td>bn</td>
</tr>
<tr>
<td>Result</td>
<td>f(a1,b1)</td>
<td>f(a2,b2)</td>
<td>...</td>
<td>f(an,bn)</td>
</tr>
</tbody>
</table>
Reduce

- A scalar result is computed by applying a commutative associative binary operator between each element in the vector.

\[ \text{Result} = 1 \odot 2 \odot \ldots \odot n \]
MapReduce

- Combination of Map and Reduce. Produces the same result as if a Map was first performed then a Reduction of the result.

Input a

\[
\begin{array}{ccc}
\text{a1} & \text{a2} & \ldots & \text{an} \\
\end{array}
\]

Input b

\[
\begin{array}{ccc}
\text{b1} & \text{b2} & \ldots & \text{bn} \\
\end{array}
\]

Result

\[
\begin{array}{ccc}
\text{f(a1,b1)} & \text{f(a2,b2)} & \ldots & \text{f(an,bn)} \\
\end{array}
\]
MapOverlap

- Similar to a Map, but each element of the result vector is a function of several adjacent elements of one input vector.

Overlap: 2
MapArray

- Variant of Map. Each element of the result is a function of the corresponding element of one of the input vectors and any number of elements from the other input vector.

```
Input a  | a1 | a2 | ... | an  
Input b  | b1 | b2 | ... | bn  
Result   | f(a1,b1,...,bn) | f(a2,b1,...,bn) | ... | f(an,b1,...,bn) 
```
Multi-GPU

- Divides the vector evenly among the GPUs
- Transparent to the user
Helpers

- Device management
- Memory management
- Testing
Example: Dot Product

```cpp
#include <iostream>
#include "skepu/vector.h"
#include "skepu/mapreduce.h"

BINARY_FUNC(plus, double, a, b, 
             return a+b;
)

BINARY_FUNC(mult, double, a, b, 
             return a*b;
)

int main()
{
    skepu::MapReduce<mult, plus>
dotProduct(new mult, new plus);

    skepu::Vector<double> v0(1000,2);
    skepu::Vector<double> v1(1000,2);

    double r = dotProduct(v0,v1);
    std::cout<<"Result: ".<<r<<"\n";

    return 0;
}
```
Evaluation

- Time Distribution
- Gaussian Blur
- Dot Product
- Runge-Kutta ODE Solver
Gaussian Blur

Gaussian Blur: one filtering

- CPU
- OpenMP
- OpenCL single
- OpenCL multi
- CUDA
Gaussian Blur

Gaussian Blur: nine filterings

- CPU
- OpenMP
- OpenCL single
- OpenCL multi
- CUDA

Time (Sec) vs Image Size (Pixels)
Dot Product

Dot Product without memory transfer time

- CUBLAS
- SkePU CUDA
- SkePU OpenCL Single
- SkePU OpenCL Multi

Time (Sec) (1000 runs) vs Vector size (# of doubles)
Runge-Kutta ODE Solver

![Graph showing the performance of different ODE solvers](chart.png)

- **ODE solver**
  - ls-seq-def
  - ls-seq-A
  - ls-shm-def
  - ls-shm-A
  - skepu-CL
  - skepu-CL-multi
  - skepu-CU
  - skepu-OMP
  - skepu-CPU
  - CU-hand

- **Time (Sec)**
  - 0
  - 200
  - 400
  - 600
  - 800
  - 1000
  - 1200

- **Problem size (N)**
  - 200
  - 300
  - 400
  - 500
  - 600
  - 700
  - 800
  - 900
  - 1000
Conclusions

- Skeleton programming is a viable approach to high-level portable GPU computing
- A general interface with multiple back ends provides flexibility to use different architectures
- Memory transfers between host and device can be a bottleneck. Lazy memory copying helps to remedy this
Future Work

- More skeletons, (Scan has been implemented after the article was written). Also task parallel skeletons.
- Two-dimensional data structure
- (Auto)-tuning (Work in progress)
SkePU is available for download at:

http://www.ida.liu.se/~chrke/skepu