Mapping Streaming Applications to Heterogeneous Platforms with Data Parallel Accelerators

GPGPU Day
28th June 2012, SARA, Amsterdam.

Ana Balevic, Bart Kienhuis
LIACS
Introduction

- LIACS - Polyhedral Process Networks (PPNs) model, long tradition of research on PPNs and automatic mapping on embedded systems and FPGAs

- Research cooperation between LIACS, Compaan B.V. and ACE Compilers on MEDEA+ European project on Tera-Scale Multi-Core Processor Architectures (TSAR)

- High performance and embedded platforms converging Heterogeneous X*CPUs + Y*GPUs Platforms
  - Embedded: TI’s OMAP (ARM+special coproc), NVIDIA Tegra
  - HPC: Lomonosov@1.3petaflops (1554x GPU+4-core CPUs)

- Mapping of applications onto heterogeneous platforms require careful parallelization and mapping with in-depth architecture considerations – can intermediate models help?
Problem Statement

- Given a sequential application how to obtain parallel code executable on a heterogeneous platform with a GPU accelerator?

Challenges:
- How to transform sequential into parallel code for execution on different components, incl. accelerators?
- How to efficiently manage communication between platform components?
- How to obtain and exploit multiple types of parallelism? (data, task, pipeline)? How to influence balance of diff. types of parallelism?
Parallel Programming SoA

Parallel Execution: Parallel Program + Target Arch. Compilers + Run-Time

Obtaining a Parallel Program:

Explicit Parallel Programming

Semi-Automatic (Languages, Directive-Based Parallelization)

Automatic Parallelization

Task + Pipeline Parallelism

Transformation frameworks

POSIX Threads

OpenMP

Intel's TBB

OpenACC

CAPS/HMPP

+run-time environments

OpenMP, TBB, StarSS, StarPU

skeleton approaches

Classical Compiler Analysis:

data parallelism – CETUS, PGI

Polyhedral Model:

task + pipeline parallelism

Compaaan/PNgen

memory model

our research

data parallelism

(LooPo, Pluto, PoCC, ROSE, SUIF, CHiLL)

(single component view)
Polyhedral Model

- Static Affine Nested Loop Programs (SANLPs)
  - Loop bounds, control predicates, array references – affine functions in loop indices and global parameters
  - Host spots - streaming multimedia and signal processing applications

- Polyhedral model of a SANLP can be automatically derived based on Featurier’s fundamental work on array dataflow analysis (see: PoCC, PN, Compaan)

```
int A[X][Y]
for (int x=0; x<= X; x++)
  for (int y=0; y<= 2*x; y++)
    tmp = A[x][y];   // Statement S
```

Example SANLP code: a for-loop nest

\[
\mathcal{D}_S = \{ \mathbf{i}_S \mid \mathbf{i}_S \in \mathbb{Z}^N, \mathbf{D}_S \mathbf{i}_S \geq 0 \}
\]

\[
\mathbf{i}_S = \{i, n, 1\}^T
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & -1 & 2 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
X \\
1 \\
\end{bmatrix} \geq 0
\]

\[
\mathcal{F}_S^A(\mathbf{i}_S) = F_S^A \mathbf{i}_S
\]
Polyhedral Process Networks

SANLP

```c
float predict(float up, float left) {
    return left - up;
}
float A[N][N];
for(i=0; i<N; i++)
    for(j=0; j<N; j++)
        P: A[i][j]=produceInitialValues(...);
for(i=1; i<N; i++)
    for(j=1; j<N; j++)
        T: A[i][j]=predict(A[i-1][j], A[i][j-1]);
for(i=1; i<N; i++)
    for(j=1; j<N; j++)
        C: consume(&A[i][j]);
```

PPN – a network of autonomous processes communicating through channels
- Polyhedral node domain (PND) – iteration space of statement’s for loop nest
- Process execute operations from PND
- Channels are result of exact dependency analysis, producer-consumer mapping
Sample Streaming Multimedia Application: M-JPEG Encoder

```c
for(frameId=0; ...) {
    initVideoIn(&hi);
    for (j = 0 ; j < VNumBlocks; j++)  {
        for (i = 0 ; i < HNumBlocks; i++)  {
            mainVideoIn(&block[j][i]);
            mainDCT (block[j][i], &block[j][i]);
            mainQ (block[j][i], &block[j][i]);
            mainVLE (block[j][i], &stream);
            mainVideoOut(hi, &stream);
        }
    }
}
```

- Polyhedral Process Network
  - Task parallelism
  - Separation of communication and computation
  - Communication via channels (FIFOs)

Automatic conversion to PPN (PNgen/Compaan)
But, what if we want...

- Data Parallelism?
- GPU acceleration?
- Efficient Host-Accelerator communication?
- To adjust token granularity?
A Structured Approach for Mapping Streaming Applications based on PPNs

Coarse-Grain Task Parallelism

Data Parallelism

Hierarchical Parallelization + Token Granularity

1

Fine-Grain Data Parallelism

KPN2GPU Mapping

2

KPN2GPU Mapping

Task Offloading and Efficient Stream Buffer Mechanism

3

To: Data Parallel

Automatic

Explicit Programming (OMP, CUDA)

Sequential

Pipeline Parallelism

Pipeline Parallelism
Hierarchical Parallelization Approach and Adjusting Token Granularity

for (frameld=0; ... ) {
    ...
    for (jb = 0 ; jb < VNumBlocks; jb++) {
        for (ib = 0 ; ib < HNumBlocks; ib++) {
            mainDCT(&blockIn[jb][ib], &blockOut[jb][ib]);
            Y1 Shift for (i=0; i<8; i++)
            Y2 Bound for (j=0; j<8; j++)
            Shift tmp[i][j] = dotP1(blockIn, c);
            U1 DCT1 for (i=0; i<8; i++)
            U2 Bound for (j=0; j<8; j++)
            DCT1 shift(&blockIn[i][j]);
            DCT2 for (j=0; j<8; j++)
            blockOut[i][j] = dotP2(tmp, c);
            Bound for (i=0; i<8; i++)
            V1 ... for (j=0; j<8; j++)
            Bound for (i=0; i<8; i++)
            bound(&blockOut[i][j]);
    }
}

STREAM NODE
PPN Domain

Coarse-Grain DP
Detection of CG Parallelism (e.g. Feautrier) and Conversion to CUDA Grid Specification

Fine-Grain DP
Conversion to CUDA Kernel Code And CUDA Block Specification via DPV/KPN2GPU
Data Parallelism: DCT Node

L2 PPN:

- Data Parallelism identification via scheduling and tiling, e.g.

- Construction of Data Parallel View on PPNs (SCOPES’11)

- CUDA kernel functionality generated from a PPN process; executed by 8x8 thread blocks

- PPN specification used to generate host+comm. code

- Multiplicity + multirate concepts leveraged to optimize communication and kernel mapping

- CUDA Streams parallel tasks (Fermi)
Asynchronous Data-Driven Execution

- PPN: naturally exposes coarse-grain task and pipeline parallelism

- Each coarse-grain PPN node is typically mapped to a single thread of execution
  - Baseline approach: PPN Process -> Asynchronous Task
  - Task Structure: {Channel Read, Compute, Channel Write}
  - Accelerator Offloading: Compute -> Stream In, GPU Kernel, Stream Out
Efficient Heterogeneous Stream Buffer Design

b) CPU Producer Thread

for (fid=0; fid<N; fid++){
    //push token in QA
    wait(buffQA->emptySlots);
    //produce/load
    token[fid]= ... 
    buffQA->put(token[fid]);
}

c) GPU Transformer Thread

for (fid = 0; fid <N; fid++) {
    //pop token from QA
    wait(buffQA->fullSlots);
    wait(buffQC->emptySlots);
    inTokenQA = buffQA->getRdPtr();
    outTokenQC = buffQC->getWrPtr();
    transformerKernel<<<NB, NT, NM, computeStream>>>
        (inTokenQA, outTokenQC);
    buffQA->incRdPtr();
    buffQC->incWrPtr();
    signal(buffQA->emptySlots);
    //init token push in QC
    buffQC->put(token[fid]);
}

d) Stream Buffer (FIFO)

for (;;) {
    waitAsyncWriteToComplete(...);
    signal(buff->fullSlots);
}
Preliminary Results - KPN2GPU

CPU vs KPN2GPU (Computation-only)

Comparison CPU (ref) vs GPU: Kernel Computation

GPU Performance Breakdown (Normalized Execution Time)

Leiden University. The university to discover.
Conclusions

- A structured approach for deriving task, data and pipeline parallel applications and mapping them onto heterogeneous platforms with data parallel GPU accelerators.
  - identification and extraction of fine-grain data parallelism in a PPN specification
  - token granularity representation and hierarchical parallelization techniques in order to leverage coarse-grain task and pipeline parallelism
  - efficient stream buffer design for CPU-GPU communication – improved accelerator offloading for streaming applications

- Implemented a Java-based KPN2GPU mapping tool as an extension to the Compaan compiler framework

- Preliminary results promising; experiments in progress – more results soon to come!

Contact: balevic@liacs.nl