Computing Without Processors
By Satnam Singh

Presented By: Divya Shreya Boddapati

Introduction
The distinction between hardware and software is being blurred.

The current iteration of mainstream computing architecture is based on cache-coherent multicore processors.
• Writing concurrent and parallel software for such systems is based on abstractions developed for writing operating systems.
• Use a diverse collection of processing elements working in concert and tuned to perform different types of computations and communication.

**Challenges to Overcome**

• To make heterogeneous computing in the cloud a reality there are many technical challenges to overcome.
  • How to virtualize GPU and FPGA computations so they can be moved to different physical resources.
  • Security violations.
  • Design of data structures that are suited for use on multicore processes and heterogeneous computing elements connected with a complex memory hierarchy or on-chip network.
Alternative processing elements such as GPU and FPGAs.

- One place we are likely to see the deployment of heterogeneous computing systems is in the cloud.
- Heterogeneous systems will need programming models and runtime infrastructures that allow the remapping of computations to different types of processing hardware.

What is Possible Today

- More than nested parallel for loops
  - To make effective use of heterogeneous computing programmers need to look beyond the problem of how to parallelize nested for loops and consider how to compute as much as possible in parallel in the highly irregular structure of modern applications.
  - Some techniques that make it easier to exploit irregular parallelism include software transactional memory (STM) and automatic mutual exclusion (AME)
• **Data-parallel computing with GPUs.**
  
  - To get good performance from GPUs, one undergoes a “Back to future” experience.
  
  - Modern GPU architectures have a memory hierarchy that needs to be explicitly programmed to obtain good performance.
  
  - Programmers are forced to program close to the machine using special GPU programming languages.
  
  - Languages like CUDA were developed for effective use of parallelism in GPU.
  
  - In future C++ compilers can target GPU and eventually FPGAs.
  
  - To extract best from GPU we still need to use CUDA that expose low-level architectural features.

• **Language-neutral, multi target data-parallel programming.**
  
  - Accelerator system from Microsoft provides certain kinds of data parallel descriptions to be written once and then executed on three different targets.
  
  - Accelerator achieves this by constraining the data types used for parallel programming and by providing a restricted set of parallel array access operations.
  
  - Not all kinds of data-parallel algorithms are a suitable match.
  
  - This abstract language is embedded in a conventional concrete language, and programs of the abstract language are expressed as data structures in the concrete language.
The task of performing an element-wise data-parallel addition of two arrays.
This program can be compiled and linked with the Accelerator library.
When executes it generates GPU code at runtime on the host processor by JITing (using just in time compilation)
JITing model hides many low-level details about GPU code generation and hardware communication.
Note that data parallel addition of arrays \( x \) and \( y \) is specified by using overloaded definition of "\(+\)" operator.

```c#
using System;
using Microsoft.ParallelArrays;
namespace AddArraysPointwiseMulticore
{
    class AddArraysPointwiseMulticore
    {
        static void Main(string[] args)
        {
            var x = new FloatParallelArray(new[] {1.0F, 2, 3, 4, 5});
            var y = new FloatParallelArray(new[] {6.0F, 7, 8, 9, 10});
            var multicoreTarget = new MulticoreTarget();
            var z = x + y;
            foreach (var i in multicoreTarget.ToArray1D (z))
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
        }
    }
}
```

To run the same computation as SSE3 vector instruction on a multicore processor we would write exactly the same program but specify different target.
To emphasize language-neutral the example in Figure above switches the source language to C# for the multicore SSE3 version.
When run it dynamically generates (JITs) SSE3 vector instructions and delivers exactly the same result as GPU version.
- Mapping explicit data-parallel computations on FPGAs

```csharp
open System
open Microsoft.ParallelArrays
let main(args) =
  let x = new FloatParallelArray (Array.map float32 [[1; 2; 3; 4; 5]])
  let y = new FloatParallelArray (Array.map float32 [[6; 7; 8; 9; 10]])
  let z = x + y
  use fpgaTarget = new FPGATarget("adder")
  fpgaTarget.ToArrayID(s)
```

- Again to emphasize the language neutral aspect of Accelerator the example above switches the source language to F#.
- The above example doesn’t write out the result of executing data-parallel description on an FPGA chip as we cannot currently support a JITing model for it.
- This target works in an offline mode and generates a VHDL hardware description that is deposited in a file. (adder.vhd)

- To implement this algorithm represent the computation with nested for loops, which use explicit array indexing to capture the notion of a sliding window.
- This is poor starting point as its difficult to map the arbitrary array indexing operations into efficient memory access operations on various targets.

- A better way is to express this computation in terms of a whole array operation combined with explicit memory-access transformations.
- The Figure beside shows an example of memory transform operation shift.
- It takes an input array x and produces an output array that may be shifted right or left.
The Figure shows an application of shift operation used to describe successively shifted versions of the input array, and then whole array multiplication is performed in parallel followed by a parallel reduction to yield the convolved output.

In Accelerator the whole array description of the convolution operation can be expressed as shown in the Figure beside.

“computeTarget” will tell which target the code is to be processed.

“a” is a regular C# array where as “x” has different type : FloatParallelArray which is a data parallel array that may live in a different address space from the host C# program.

Sequential version was run on a computer with 2 Intel Xeon E7450 processors, each clocked at 2.4 GHz. Machine had 32GB of memory and ran Windows Server 2008 R2.

The Accelerator version was run using the SSE3 multicore target on same machine, testing 2 different graphic cards: ATI Radeon HD 5870 and an Nvidia 470 GTX using the DX9 target.

To understand the performance improvement the Accelerator version of two-dimensional convolver computation is compared against a sequential version. Consider the diagram beside.

It works by applying two one dimensional convolutions and takes as parameter the specific target to be used for the data parallel execution.
The Importance of Specifying communication

Shift is a powerful memory-access abstraction which reveals what information to reuse so its worth caching, storing in registers or placing into local shared memory on a GPU.

When FPGA is used to implement the convolver, the shift operations are used to infer delays that allow us to reach each data item once but use it three times. Consider the diagram beside.

The above diagram is a good fit for FPGA technology because of abundance of registers for implementing pipeline delays.

Multiple Targets.

We want a system that compiles a single data-parallel description onto a mixture of processing elements.

The compilation process would also generate the software needed to orchestrate the movement of data between the different types of hardware elements and coordinate their execution.

What we might see in 18 Months

Turing programs into circuits has provided digital designers with merely the ability to write highly stylized circuit descriptions that are heard beat away from low-level register transfer-level description.

These techniques do not form the basis of an approach that converts realistic software into hardware circuits.
• Encouraging projects working on this aspect are
  • Liquid Metal project at IBM, based on Lime language that extends Java with special concurrency constructs.
  • Kiwi project at the University of Cambridge by David Greaves and at Microsoft Research Cambridge (which works on unmodified standards .NET programming languages and relies on multithreaded programs to describe important parallel execution economics)
  • Intel’s Sandy Bridge processors have AVX which provide data-parallel operations over 256-bit registers.
  • AMD’s Fusion program promises ever tighter integration between CPUs and GPU cores on the same die.
  • There is already a proof-of-concept single-package chip that combines Intel Atom processor and Altera FPGA.

• Further Out
  • There has been progress in C language that manipulates heap data structures with pointers into circuits using shape analysis and region types.
  • This ability to convert an algorithm to preserve its meaning but refract its work over a different representation of its data makes it appropriate for programming of heterogeneous systems possible.
Conclusion

• As demand for performance and energy consumption increases the programmers are forced to opt for heterogeneous systems.

• It is possible to identify a constrained class of problems that share some key architectural assumptions.

• The Accelerator system is an example of this approach.

• By combining domain specific languages in mainstream programming languages we can develop mechanisms that make the programming of heterogeneous systems tractable.

Questions?
KEEP CALM
U PEOPLE WERE
AWESOME THROUGHOUT
THE PPT!!! THANK YOU