The Sisal Model of Functional Programming and its Implementation

Jean-Luc Gaudoit et al.

Presented by: Kyle King
October 2, 2007
Overview

- Motivation
- The Sisal Language
- Shared memory compiler
- Distributed memory compiler
- Sisal on multithreading architecture
Motivation

- C/MPI is hard
- Lack of programming quality in parallel programs. We want:
  - Maintainability
  - Extensibility
  - Portability
  - Simplicity
- Make the compiler do all the work
The Sisal Language

- Stream and Iteration in a Single Assignment Language
- A Functional Language
  - No side effects
  - No implicit state
  - Everything is a copy
  - Single Assignment (No writes ?!)
define Main

    type Info = array[ integer ]

    function Main ( Data : Info returns Info )

        function Split (Data : Info  returns Info, Info, Info )
        for E in Data
            returns array of E when E < Data[ 1 ]
                array of E when E = Data[ 1 ]
                array of E when E > Data[ 1 ]
        end for
    end function

    % routine body
    %

    if array_size( Data ) > 2 then
        let
            L, Middle, R := Split( Data )
        in
            Main( L ) || Middle || Main( R )
        end let
    else
        Data
    end if

end function % quicksort
OSC

- Optimizing Sisal Compiler
- Shared memory
- Performs extensive analysis
- Dataflow is explicit
- Implicit parallelism
- Explicit sequential code
Shared Memory Compiler Optimizations

- Everything is a copy
- When do we free?
  - Reference counts
- Copy Elimination
  - Analysis can remove lots of references
  - Once assignments are done being made, we can reuse the containers since there are no pointers into the data
Shared Memory Compiler Optimizations

- Build in place
- Programs often compute left, right, inner values separately then merge
- This incurs a lot of overhead in functional languages
- Allocate an extra buffer the size of the array (since sizes are dynamically predetermined) and build in place
Shared Memory
Compiler Optimizations

- Other Optimizations
  - Analysis makes vectorization easy
  - Loop Fusion
  - Double Buffering Pointer Swap
  - Inversion
Distributed Memory Compiler Optimizations

- **D-OSC**
- The master process divides loops evenly amongst the slave processes
- **Activation Record Queue**
  - Record may contain parallel code
  - Become a master!
  - Each process has a listener
Distributed Memory Compiler Optimizations

• Rectangular Array
  • One descriptor => Faster lookup
  • Not so good in practice
    • No sub-arrays to share
    • Requests from other processes tend to be symmetric to the original structure of the array
Distributed Memory Compiler Optimizations

- Block Messages
- Multiple alignment
  - Finds “affine” loops
  - Passes one large array instead of multiple passes of individual elements
- Decreases communication overhead
Multithreaded Execution

• Asynchronous

• Based on the availability of data

• Thread firing models

  • Blocking

  • Non-blocking
Threads

• Each thread has
  • A block of code they execute
  • A synchronization number
  • A synchronization unit to send results they to
Threads

Blocking
• Instantiated when code block is reached
• Executed when synchronization number reaches zero
• Paused as needed

Non-Blocking
• Instantiated when all inputs available
• Executes until completion
Multithreaded Execution

- How do threads get data from other processes?
  - Single-phase: Request the data, wait for the data, continue execution
  - Split-phase: Request the data, die, allow another thread to continue execution
- Blocking threads use both, single for local access, split-phase for remote access
- Non-blocking threads only use split-phase
Performance

- The performance of the blocking vs. non-blocking thread models is completely dependent on the memory layout.
- Special storage schemes can be designed using non-blocking that take advantage of locality.
- Blocking has a higher cache miss rate.
Current Status

• Versions exist for the Cray X-MP, Y-MP, 2; Sequent, Encore Alliant, and others.

• Lawrence Livermore National Laboratory has cancelled the project

• Lives on at SourceForge
Questions?