LLVM backend for TILE64

The backend in its current state provides the following features:

• Tilera ABI compliance
• Generating position-independent code
• Handling variadic functions
• Utilizing TILE64 VLIW capabilities
• Allocating dynamic local variables
• Generating Tile Processor Assembly Code

It is available on GitHub:
https://github.com/llvm-tilera

What is TILE64?

TILE64 is an energy-efficient massively parallel processor architecture, which was the first commercial product of Tilera Corporation.

• 64 general purpose processor cores (tiles) connected by a mesh-network
• short-pipeline, in-order, three-issue cores
• VLIW instruction set
  – supporting RISC instructions
  – extended with SIMD and DSP-related operations
• the speed of the interconnection between tiles is one hop per tick
• the edges of the mesh are connected to I/O interfaces
  – four DDR2 controllers
  – two 10-gigabit Ethernet interfaces
  – two four-lane PCIe interfaces
• a software-configured “flexible” I/O interface

Unresolved issues

There are issues seeming to be special ones, but trends in hardware industry make them common problems for new architectures. Such two issues are the followings:

• There are issues seeming to be special ones, but trends in hardware industry make them common
• Problems for new architectures. Such two issues are the followings:
  • There are issues seeming to be special ones, but trends in hardware industry make them common
  • Problems for new architectures. Such two issues are the followings:
    • There are issues seeming to be special ones, but trends in hardware industry make them common
    • Problems for new architectures. Such two issues are the followings:
      • There are issues seeming to be special ones, but trends in hardware industry make them common
      • Problems for new architectures. Such two issues are the followings:
        • There are issues seeming to be special ones, but trends in hardware industry make them common
        • Problems for new architectures. Such two issues are the followings:
          • There are issues seeming to be special ones, but trends in hardware industry make them common
          • Problems for new architectures. Such two issues are the followings:
            • There are issues seeming to be special ones, but trends in hardware industry make them common
            • Problems for new architectures. Such two issues are the followings:
              • There are issues seeming to be special ones, but trends in hardware industry make them common

Exploiting the power of multiple cores

Forced by the lack of a platform-independent model for thread-level parallelism in LLVM, one needs to use architecture-specific library calls to implement parallel applications.

Unchangeable default behaviors

Some aspects of code generation are hard-coded into LLVM CodeGen library. The choices that have been made there may fit mainstream architectures, but just don’t meet the needs of other, more specialized ones.

Argument passing according to the Tilera ABI

• First ten words are passed in registers
• Further parameters are passed in the argument space of the caller
• Alignment is forced by padding on the stack and even in registers
• Padding is conserved inside structs during passing
• All parameters are passed entirely either in registers or on the stack
• If a return value cannot fit in ten registers, then it is passed indirectly

Argument passing by LLVM CodeGen

Some aspects are hard-coded into the library:

• Struct arguments are broken up into component values
• Values are split into register-sized parts
• Alignment is forced by padding on the stack and even in registers
• All parameters are passed entirely either in registers or on the stack

Resolving the discrepancy

Hard-coded decisions can be worked around by restricting the way LLVM IR is used. Constraints on the usage of struct values could be forced by a transformation pass preparing for instruction selection.

Transformation for passing a struct value as argument:

Transformation for passing an undersize struct value as result:

VLIW packetizing

Utilizing VLIW feature introduced in LLVM 3.2 gives promising results, however the machinery could be improved to fully suit VLIW capabilities of TILE64.

Implementing a VLIW packetizer

VLIW-scheduler automaton generated from TableGen description:

• Declare functional units
• Classify instructions regarding functional units
• Bundling by looking for a suitable functional unit
TILE64 has further constraints for valid bundles:

• three functional units
• 64-bit bundles with different sized instruction codes
• therefore, there are two- and three-issue bundles

Constraints beyond availability of functional units cannot be expressed by actual TableGen facilities. A patched VLIWPacketizerList is used for validating bundles created by the scheduler automaton.

Measuring execution times

We use pseudo-instructions to postpone generating actual code until proper information is available. Most of these instructions are suitable for fully automatic generation.

However, custom printing mechanisms have to be provided sometimes. One such example is the generation of position independent code:

Nasty technical details

The EITKIC_EU-2012-0001 project is supported by the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fund.

David Juhász, Tamás Kozsik
juha@caesar.elte.hu, kto@elte.hu
Department of Programming Languages and Compilers
Faculty of Informatics, Eötvös Loránd University
Pázmány Péter sétány 1/C, Budapest 1117, Hungary