Parameterization of Compiler Optimizations For Empirical Tuning

Qing Yi
University of Texas at San Antonio
Empirical Tuning of Compiler Optimizations(1)

- Approach 1: use compiler command-line options
  - Compilers can be configured in many ways
    - What strategies to use, which optimizations to apply...
  - Command-line options offer only a few knobs
    - -O1 -O2 -O3 -fast, -g, ...
  - How about different strategies for different fragments?
Empirical Tuning of Compiler Optimizations (2)

- Approach 2: use program annotations
  - Can specify many different transformations
    - What strategies to use, which optimizations to apply, ...
  - Allow different strategies for different fragments
  - Problem: how smart is the search engine?
    - Dependence, memory, register pressure, ...
    - Needs ability to perform program analysis
Approach 3: combine compiler and search engine

- Compiler knows about the program
  - What strategies to use, which optimizations to apply, ...
  - Dependence, memory, register pressure, ...
- Problem: flexibility and composibility
  - Compiler must be shipped together with application
  - Compiler must know how to exploit the search space
  - The compiler writer decides everything
Empirical Tuning of Compiler Optimizations

- **Approach 4: Parameterization of optimizations**
  - Compiler generates parameterized output
    - What strategies to use, which optimizations to apply, ...
  - Search engine exploits the configuration space
    - Use information from the compiler
    - Dependence, memory, register pressure, ...
  - Code generator generates program executable
    - Applies configuration to parameterized code

Diagram:

- Application → Optimizer
  - Configuration space + constraints
  - Parameterized code
  - Search Engine
    - Configuration
  - Code generator
    - Executable
  - Machine
    - Result
    - Profile information
Parameterization of Optimizations---

**Blocking**

\[
\begin{align*}
&\text{do } _i = 1, N, bi \\
&\quad \text{do } _j = 1, N, bj \\
&\quad \quad \text{do } _k = 1, N, bk \\
&\quad \quad \quad \text{do } i = _i, \min(N, _i+bi) \\
&\quad \quad \quad \quad \text{do } j = _j, \min(N, _j+bj) \\
&\quad \quad \quad \quad \quad \text{do } k = _k, \min(N, _k+bk) \\
&\quad \quad \quad \quad \quad \quad \quad C(i,j) = C(i,j)+A(i,k)*B(k,j)
\end{align*}
\]

- **Parameter: blocking sizes**
- **Configuration space** bi(1..N) * bj(1..N) * bk(1..N)
  - Tunable at both installation and run time
Parameterization of Optimizations---

Unrolling

\[ \text{do } i = 1, N, bi \]
\[ \text{<repeat } \#i:i=>i+bi> } \]
\[ C(\#i,j) = C(\#i,j) + A(\#i,k) \times B(k,j) \]
\[ \text{</repeat> } \]

- Parameter: unroll size
- Configuration space bi
- Need code generator to produce executable
  - Not tunable at runtime
Work in progress---Challenges

- Not all transformations can be parameterized
  - Loop fusion, loop interchange, scalar replacement, memory reorganization,…

- Transformations interact with each other
  - Exponential combinations of configurations
  - Interactions may not be parameterizable

- How to encode program analysis results from compiler
  - Dependence constraints, insight about programs
  - Information useful to the empirical search engine
Current status

- Loop transformations
  - Parameterization of loop fusion, unrolling, interchange
  - Tuning at installation time and at runtime

- Collaborations
  - Rice university
  - LLNL
  - U. of Tennessee at Knoxville
  - U. of Texas at San Antonio