GLIMPSES: Memory and program behavior estimation for SPEs
Jaswanth Sreeram, Ling Liu, Santosh Pande
Motivation

“Prototyping large codebases for porting to SPEs is challenging”

“Need a way to quickly evaluate program behavior and its suitability for SPEs” – Important for legacy code/reuse
Motivation (contd)

• Porting large codebases to SPEs is challenging
  – Limited local store
  – High branch penalty
  – Geared towards vectorizable code
  – Code/data partitioning is not trivial
  – SPE-SPE, SPE-PPE interactions

• Provide programmer with tools to
  – Understand dynamic program behavior
  – Quickly construct candidate partitions for SPEs
  – Evaluate/Quantify partitions’ suitability for SPEs
GLIMPSES: Tool Overview

• Dynamic Call Graphs

• Memory Requirements
  – Dynamic
  – Analytical

• Memory Access Patterns
  – Locality (spatial, temporal, neighbor affinity)

• Partitioning
  – Criteria based estimates

• Visual, interactive
Dynamic Call Chains

Graph Visualization Area

Results Display Panel
Call chains...contd
Mpeg-2 Decode

- Zoom view
- Shows dynamic call chains for a program run (in this case the program is mpeg2-decode)
GLIMPSES

C/C++ program

LLVM compiler flow

Bytecode

Analysis & Instrumentation Passes

Instru. Bytecode

Link

Runtime

Test Inputs

Execute

Profile Trace

Analytical Memory Estimator

Partition Estimator

Dyn. Memory Estimator

GraphML Trace

Visualization Engine

Test Inputs

Execute

Profile Trace
Memory Behavior

• Estimate static and dyn. memory usage
  – Code, stack and heap (per function)
  – Usage < SPE LS limit ?

• Estimate function attributes
  – Branch density
  – Number of Auto-vectorizable loops

• Analytical estimation
  – Detect program objects affecting dynamic memory behavior
  – Show correlation between these program objects and memory usage.
    • Construct an arithmetic expression for amount of memory allocated, in terms of inputs or other program objects
Analysis Estimator: MPEG2 Example

Code segment

```
for (……)
{
    if (cc==0)
        size = Picture_Width*Picture_Height;
    else
        size = Chroma_Width*Chroma_Height;

    if (!(backward_reference_frame[cc] =
        (unsigned char *) malloc(size )))
        Error(…);

    if (!(forward_reference_frame[cc] =
        (unsigned char *) malloc(size )))
        Error(…);
}
```

Result

```
__Malloc_size__1 =  1024

__Malloc_size__2 =  0 +
                    Coded_Picture_Width*Coded_Picture_Height

__Malloc_size__3 =  0 +
                    Coded_Picture_Width*Coded_Picture_Height

__Malloc_size__4 =  0 +
                    Coded_Picture_Width*Coded_Picture_Height

__Malloc_size__5 =  0 +
                    Chroma_Width*Chroma_Height

__Malloc_size__6 =  0 +
                    Chroma_Width*Chroma_Height

__Malloc_size__7 =  0 +
                    Chroma_Width*Chroma_Height

__Malloc_size__8 =  0 +
                    Chroma_Width*Chroma_Height
```
Memory Access Patterns

- Locality metrics for loads/stores
  - Spatial Locality
    - “Loads to different addresses in a spatial window”
  - Temporal Locality
    - “Loads to same address in a time window”
  - Neighbor Affinity
    - “Loads to addresses within a space and time window”
Locality Measures (mpeg2decode)

Memory Access Locality:
Recurrence of Loads

Number of loads

Number of times recurred
Locality measures: Affinity

Locality: Neighbor Affinity

Number of Loads

"NA" Values
Program Partitions

- Provide programmer with possible partition candidates
  - Can be based on criteria:
    - Memory consumption
    - Memory reference behavior
    - Branch density
    - Auto-vectorizable loops
    - Aliasing
    - Combination (a “rank” metric)

- Does not assume code/data overlays
Partitioning

• Start with earliest leaf node in dyn. call graph in a partition

• Try to add its parent to the partition
  • Try to add all of parent’s children to the partition
  • If they can be added, try to add parent to partition.

• Try to add parent’s parent to partition

• Estimates only: No code generation

• Programmer to take care of “cloning”.

• Can produce interprocedural, context sensitive alias information.
  • Given two partitions, can they alias each other’s data?
Status

• Several features/improvements planned
  – Alias Analysis information for refining partition-set
  – Alias Analysis information for data pinning/prefetching opportunities.
  – Leverage DataStructureAnalyses for smart memory allocation on SPUs

• Tested on
  – Workloads from SPECINT
  – Workloads from mediabench
  – ODE (Open Dynamics Engine)

• Beta version to be released shortly.
The End

Email contact:

jaswanth@cc.gatech.edu