Article Title

Archit Gupta, Sohum Datta
University of California, Berkeley

Branch divergence in GPGPUs has attracted a lot of interest in the computer architecture research community. Various solutions have been suggested to mitigate divergence in SIMD architectures. However, a lot of these branches are artifacts of mapping of an algorithm to a GPU and are not intrinsic to the algorithm. In this project, we will analyze the nature of branches that a GPU encounters and classify these branches according to their intrisicness to the underlying algorithm. Most solutions rely on some assumptions regarding the nature of branches encountered by the GPU, which might not be entirely true if the extrinsic branches account for most of the performance degradation

Introduction

Graphics processor units are now being extensively used for scientific computation because of tha availability of a large number of parallel resources. However, since the shader cores were not originally designed for general purpose computing, they perform poorly on code with a significant amount of control flow.

Additionally, unlike vector processors, most GPUs do not have a tightly coupled scalar core. So, in order to map a general scientic problem to a GPU, the scalar part of the program can be processed in one of the two ways:

1. Processing the scalar part of the program on the CPU and the parallel part on the GPU.
2. Serial code can be executed on the GPU as part of the execution kernel.

While the former method involves significant amount of data movement (owing to the fact that the CPU and GPU have seperate memory sub-systems, the latter approach causes the inclusion of a significant amount of control flow to the existing kernels.

Motivation

Figure 1 shows a code-snippet of the breadth first search (bfs) algorithm from the parnoil benchmark suite. One can see that the actual algorithm involves only the *visit_node* function to be called for each node in the graph in some order. However, in order to map the code to a GPU, additional code has been added. This adds a significant amount of control flow to the program.

More importantly, the behavior of branches in this part of the program is significantly different from the branches that are intrinsic to BFS. If these branches contribute significantly to the performance degradation, we can easily come up with solutions specific to these branches which can, in turn, significantly improve the performance of the GPU

Realted work and Our approach

Logistics and Infrastructure

In order to classify the branches in a GPU code, we intend to use GPGPU-SIM**??**, which accurately simulates the execution of a GPU. We use the PARBOIL benchmark suite**??**. The suite represents a variety of scientific applications which are commonly accelerated using GPUs today. Besides the diversity of applications in the suite, it also provides us with

```
__global__ void
BFS_kernel( ... )
{
  if(threadIdx.x < NUM_BIN){
    q.reset(threadIdx.x, blockDim);
  }
  __syncthreads();

  tid = blockIdx.x*THREADS_PER_BLOCK
                  ... + threadIdx.x;
  if( tid < no_of_nodes)
  {
    visit_node(q1[tid], threadIdx.x
        & MOD_OP, q, overflow,
        g_color, g_cost, gray_shade);
  }
  __syncthreads();

  if(threadIdx.x == 0){
    ...
    shift = atomicAdd(tail,tot_sum);
  }
  __syncthreads();
}
```

Figure 1: *Code snippet from an implementation of Breadth first search algorithm in CUDA*

base implementations of the same algorithms (which are CPU implementations) that help us identify the branches that are inherently intrinsic to a program.

In order to expand the set of applications (if required), we can also evaluate Rodinia benchmark suite**??**. This, however, does not come with a *base* implementation of each of the algorithms, so, it would require manual identification of branches.

References

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.