THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Extracting Parallelism in Mobile GPUs/CPUs with Combinatorial Architecture

Ganapathy Mani[1], Simon Berkovich[2], Duoduo Liao[3]
Department of Computer Science, George Washington University, Washington, DC USA[1,2]
Computing for Geospatial Research Institute, Reston, VA USA[3]
gans87@gwmail.gwu.edu[1], berkov@gwu.edu[2], dliao@gwmail.gwu.edu[3]

ABSTRACT

Current mobile GPU technologies are still relatively immature and require substantial improvements to enable wireless devices to perform the complex graphics-related functions. Traditional way of extracting parallelism requires more processor cores and complex programming. But mobile devices are not designed to hold many processors or execute heavy-computing program. We propose a new combinatorial architecture based on balanced incomplete block designs (BIBD) that is more suitable for mobile GPUs/CPUs, i.e., mobile heterogeneous computing, with limited resources and relative greater performance. This combinatorial architecture that has a unique property—multi-core running on a sequential code. This architecture can be used for both mobile CPUs and GPUs. Some minor adjustments to a regular compiler are needed for loading the instructions.

INADEQUACIES OF EXISTING METHODS

To increase performance, processor manufacturers extract parallelism through multi-core systems. Besides several complications, this requires tremendous efforts for organization special software for parallel processing.

PAIRWISE-BALANCED BLOCK DESIGNS

The block design is a famous combinatorial structure, which is a subset satisfying some conditions. In a processor, partitions designed by these block designs allows concurrent processing of data independent instructions in executable sequential code because the partitioning is done at the compiler-level [1]. A balanced incomplete block design on a set of v elements is a collection of b k-subsets such that each element appears exactly in r subsets and each pair of elements appears exactly in λ subsets. The balanced incomplete block design is represented by its parameters in standard notation: (b, v, r, k, λ)-configuration.

As an example of this configuration, consider a system of seven 3-element subsets of 7-element set $Z = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$ which constitutes a balanced incomplete block design of (7, 7, 3, 3, 1)-configuration (Figure 1). In this configuration, there are 7 subsets (blocks) are available each with three elements:

$$PB_1 = \{X_1, X_5, X_7\},$$
$$PB_2 = \{X_1, X_2, X_6\},$$
$$PB_3 = \{X_2, X_3, X_7\},$$
$$PB_4 = \{X_1, X_3, X_4\},$$
$$PB_5 = \{X_2, X_4, X_5\},$$
$$PB_6 = \{X_3, X_5, X_6\},$$
$$PB_7 = \{X_4, X_6, X_7\}.$$

Let's consider that there is an interaction between two objects or threads X_1 and X_6. We can find both of these elements in PB_2. Instead of transporting data from one block to another, the processing can take place instantaneously. As the result, corresponding processing blocks can be updated independently and simultaneously without interfering with others and progress in parallel. Thus this architecture works on sequential code while extracting parallelism. Other possible designs are (13, 13, 4, 4, 1), (21, 21, 5, 5, 1), (26, 13, 6, 3, 1), (35, 15, 9, 3, 1).

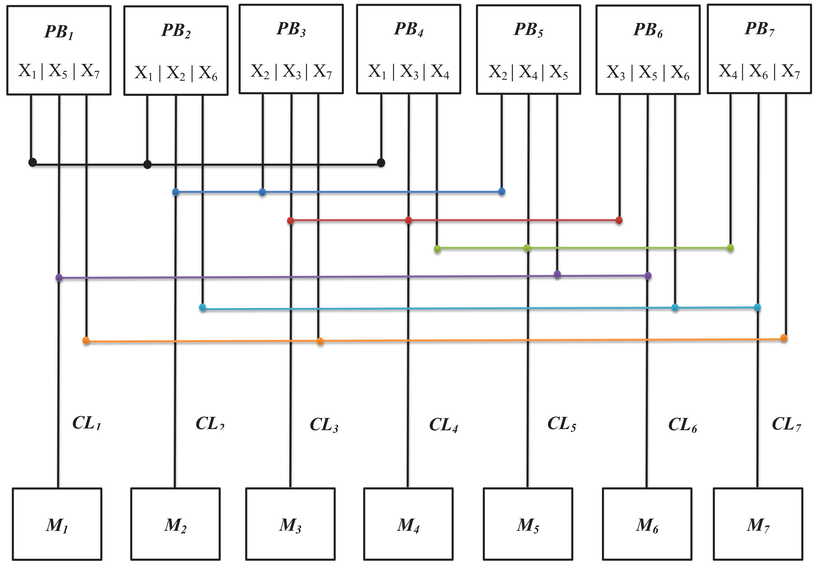

Figure 1 (7, 7, 3, 3, 1) configuration of combinatorial architecture using balanced block design: PB_i is processing block (or processing element), CL_i is communication link, M_i is memory module and X_i can be a storage element (register) or an object or a thread.

The given architecture works perfectly for a 2-operand machine since every input pair is subjected to be processed instantaneously and progress in parallel. For three or more operands, some minor adjustments are needed for the complier.

PROCESSING OF "*MULTADD*" INSTRUCTION

Consider an example of 3-operand ***multadd*** instruction—an important instruction for mobile GPU:

multadd(a, b, c);

multadd instruction multiplies first two variables and adds the third variable to that result. There are two ways to exploit the combinatorial architecture,

- Through complier, we can assign the variables to the explicit triples (storage elements) in the processing blocks as $X_1 = a$, $X_3 = b$ and $X_4 = c$. Now the operation can be performed instantly in a single processing block PB_4.
- Or through compiler, we can allocate variables randomly to any processing block and compute the ***multadd***. For example, lets assign variables as $X_1 = a$, $X_2 = b$ and $X_3 = c$,
  1) X_1, X_2 are available in PB_2. $X_1 * X_2$ result will be stored in X_6
  2) X_6 and X_3 are available in PB_6 —both can be added instantaneously—It has less data-dependency than other processors

Hence if there are 3 or more operands we can always utilize this architecture by making some minor changes to the compiler.

EXPERIMENTAL RESULTS

Our uniprocessor software simulation showed the speed increase of over 30% compared to a regular processor. Our implementation of this architecture in FPGA using configurable logic blocks (CLB) showed up to 27% speed increase with the significant reduction in transportation energy [2].

OUR CONTRIBUTIONS

- Our scheme offers intrinsic reliability with automatic fault-tolerant as well as increased availability of the data objects with the convenient control of concurrent objects.
- Since our design can be used at the instruction level, this combinatorial scheme can be adapted to different type of systems with more than two operands and several extensions are possible.
- Our processor design does not depend on centralized clock since components can operate in their own local clock coherently.

DISCUSSION AND FUTURE WORK

We plan to build up our own combinatorial design having all possible elements for 3 or more operands. We will also consider supplementing ordinary pairwise BIBD with some restricted facilities of 3-operand instructions.

REFERENCES

[1] S. Berkovich, E. Berkovich, "Methods and apparatus for concurrent execution of serial computing instructions using combinatorial architecture for program partitioning", US Patent No. 5,619,680, issued 8 April 1997.
[2] G. Mani, S. Berkovich, I. Mihai, "A Combinatorial Distributed Architecture for Exascale Computing", In 4th IEEE International Conference on Advanced Computing, 2012, pp. 1-5.