OS02: Information and Communication Technology

GPU-accelerated Branch-and-Bound Algorithm for Scheduling of Virtual Machines in Cloud datacenters

Nguyen Quang-Hung, Duy Luong, Nam Thoai, Nguyen Thanh Son

Faculty of Computer Science and Engineering, University of Technology, VNU-HCM, Vietnam

Abstract

In this paper, we consider the problem of placement of virtual machines (VMs) in a cloud datacenters. Each VM requires a fixed number of processors (cores) andcomputing power. Each host has maximum capacity resources on processors and each core has limited ofcomputing power. Each placement of a VM onto a physical machine has an estimated cost (the cost may be variable from host to host). We want to minimize the total cost of the placements of VMs. We propose here a Branch-and-Bound (BnB) algorithm that is adaptively implemented in General Purpose Graphics Processing Unit (GPGPU) and CUDA framework. The proposed BnB has heuristics to bound and prune the huge search tree effectively on GPU-side processing, and copy massive nodes from/to host to/from device. In our experiments with NVIDIA® TESLA™ M2090 GPU, the GPU-accelerated BnB implementation has good speedup in comparing to a serial BnB implementation on CPU on same input problem size.

1. INTRODUCTION

Cloud computing has become more popular in provision of computing resources under virtual machine (VM) abstraction for cloud users to run their applications.We consider problem of placement of virtual machines in the cloud datacenters.

Given a set of n virtual machines $\{VM_j(pe_j, mips_j) \mid j=1,...,n\}$ to be placed on a set of m physical machines $\{M_i(PE_i, MIPS_i) \mid i=1,...,m\}$. Each virtual machine VM_j requires fixed pe_j processors (cores) and $mips_j$ MIPS. Each host has maximum capacity resources on PE_i processors (cores) and each core has $MIPS_i$ MIPS. Each placement of virtual machine onto a physical machine has an estimated cost (T_{ij}) (the cost may be variable from host to host). We formulate the scheduling problem as following:

$$Minimize\ F = \sum_{i=1,j=1}^{i=m,j=n} x_{ij} * T_{ij}$$

Constraint 1: Total of allocated MIPS for assigned VMs is less than or equal to total maximum capacity of each host:

$$\sum_{j=1}^{n} x_{ij} * mips_j * pe_j \leq PE_i * MIPS_i\ , \forall i = 1,..,\mathrm{m} \quad (1)$$

$$x_{ij} * pe_j \leq\ PE_i\ , \forall j = 1,..,n, \forall i = 1,..,m \quad (2)$$

Constraint 2: Each j-th VM is assigned onto just one physical machine:

$$\sum_{i=1}^{m} x_{ij} = 1\ , \forall j = 1,..,n; \forall i = 1,..,m \quad (3)$$

Binary: $x_{ij} \in \{0,1\}$, here x_{ij}= 1 iff the VM_j is placed on the M_i.

We want to minimize F - the total of the placements of virtual machines. Using serial branch-and-bound (BnB) program to solve the problem is time-consuming. Therefore, we propose here a BnB algorithm that is adaptively implemented in General Purpose Graphics Processing Unit (GPGPU) and CUDA SDK framework.

In this paper, we proposed implementation of theBnB on GPU with CUDA framework that has some heuristics to: (i) bound and prune the huge search tree effectively on GPU-side processing; and (ii) considering transferring time on massive nodes from/to host to/from device. In our benchmarking with NVIDIA® TESLA™ M2090 GPU, the GPU-implemented BnB has speedup from x33 to x135 in comparing to a serial BnB implement on CPU on input problem size (nxm, n is number of virtual machines, m is number of physical machines) change from 20x20 to 100x20.

2. RELATED WORKS

A few previous works that have proposed using GPU in the branch-n-bound algorithm implementation that using a hybrid model of CPU and GPU to solve some

classical optimization problems (e.g. Knapsack, Flow shop scheduling). Lalami & El-Baz (2012) proposed an implementation of the branch-n-bound (BnB) using a hybrid model of CPU and GPU to solve a Knapsack problem. Chakroun, et al. (2013) have also proposed an implementation of the branch-n-bound (BnB) using hybrid of multi-core CPU and GPU to solve the Flow-shop Scheduling Problem (FSP). There is missing of an implementation of BnB on GPGPU for placement of virtual machines in cloud datacenter (Quang-Hung et al. 2013) with consider on real constraints such as number of cores, computing power on each core, etc..

Using GPU to speedup the performance of simplex method is proposed by few previous studies. Spampinato & Elstery (2013) proposed a revised simplex method and interior point method to solve the linear programming. Authors claimed the pros and cons of the two methods and choosing the revised simplex method to implement on two programs: serial and parallel (GPU). Results show that the GPU program is faster than the serial program.

3. BRANCH-AND-BOUND IMPLEMENTATION ON CPU-GPU FOR VIRTUAL MACHINE ALLOCATION

3.1. Example of the placement of virtual machines problem

Given the following data as input of the BnB:

- There are four types of hosts with {2, 4, 16, 2} of maximum cores and each core is {2660, 2993, 2660, 2660} Millions Instructions Per Second (MIPS).

- There are three types of virtual machines with required cores is {1, 3, 1}, and each required core needs {2500, 2000, 1000} of required MIPS.

Table 1: Cost of each placement of virtual machine onto a host

Cost	H_0	H_1	H_2	H_3
VM_0	19	34	20	31
VM_1	41	87	49	32
VM_2	8	89	97	29

We show a search tree that is generated and explored by BnB algorithm for the placement of virtual machines on physical machines (hosts) as in Fig. 1.

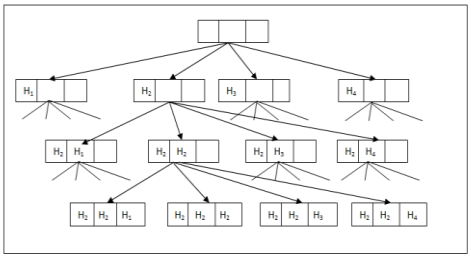

Figure 1: A search tree is generated and explored by BnB algorithm for the placement of virtual machines on physical machines (hosts)

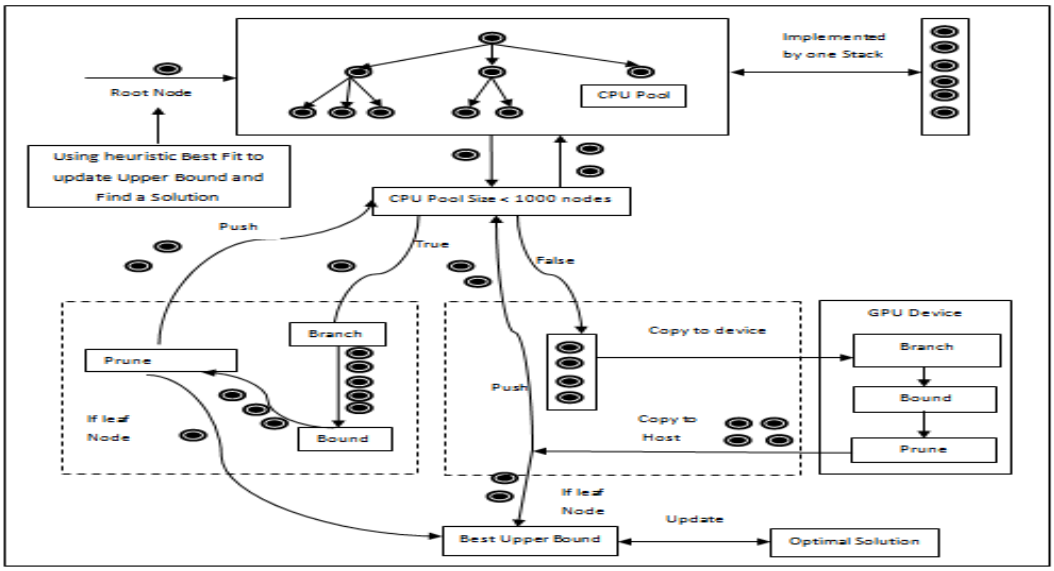

Figure 2: Brand-and-Bound (B&B) algorithm implementation on GPGPU

3.2. GPU-accelerated BnB:

The idea of this algorithm is at the same time provide a certain amount of nodes are located in the CPU pool, copy and compute them in the GPU, those nodes will generate many child nodes in branching method, then the child node will be compute cost, estimate lower bounds and prune. Promising child nodes will be push back to CPU pool to continue the loop. The loop will finish when there is no node in CPU pool. If the size of the list nodes is small, it is not efficient to launch the branch and bound computation kernels on GPU since the GPU occupancy would be very small and computations on GPU would not cover communications between CPU and GPU, so we apply the algorithm proposed hybrid model between CPU-GPU as shown in Figure 2. In this study, GPU kernels are launched only when the size of the nodes is greater than 1000 nodes. We have noticed that this condition ensures 100% occupancy for the GPU. Besides that, we use Best-fit heuristic to find an upper bound (UB) and a solution that has cost nearly the optimal solution to reduce prune times.

Branching

Each *thread* will represent a parent node that is taken from the CPU, each parent node in this kernel will generate *m* child nodes corresponding to case: set up the virtual machine (*k*+1)-th respectively on hosts from 1 to *m* (where *k* is the level of the parent node) see Figure 1.

```
Branching:
    Int idx = blockIdx.x * blockDim.x + threadIdx.x;
    IF (idx<GPU_Pool_Size)  THEN
        FOR (inti=0; i<m; i++)
        //Child node will be stored in (idx*m+i)-th address
        write(idx*m + i, promising_nodes_pool);
        END FOR
    END IF
```

Figure 3: Branching kernel on GPGPU

Bounding

Since each node in the GPU will generate *m* child nodes in the branching function so the number of child nodes will be generated equal m times of number of nodes copied to GPU. The task of this kernel are each *thread* will represent a child node which will be calculated cost and estimated the lower bound of the branch of that node is standing, the algorithm illustrated in Figure 4.

```
Bounding:
    Int idx = blockIdx.x * blockDim.x + threadIdx.x;
    IF (idx<GPU_Pool_Size * m) THEN
        cost[idx] = Compute_Cost_Node(idx);

        LB'[idx] = Estimate_Lower_Bound(idx);
        IF (LB'[idx] > UB) THEN
            eliminate(idx, promising_nodes_pool);
    END IF
```

Figure 4: Bounding kernel on GPGPU

Pruning

GPU will give the number of nodes that each node can reach the optimal solution after comparing it with the *UB* (see Figure 5). Address of promising nodes will be put into array "*outArray*" to copy back to the CPU, variable "*num_nodes_out*" for counting the number of promising nodes. We use two CUDA methods that are available for processing synchronization among threads in the GPU: "*atomicAdd*" for putting promising nodes into "*outArray*" array and "*atomicMin*" for updating the newest *UB*.

```
Pruning:
    idx = blockIdx.x * blockDim.x + threadIdx.x;
    IF (idx<GPU_Pool_Size * m) THEN
        check = Check_Node_Is_Sastified(idx);
        IF ((check) && (cost[idx] < UB)) THEN
            register int i= atomicAdd(num_nodes_out, 1);
            outArray[i] = idx;
            IF (Node(idx).level == leafNode_level) THEN
                atomicMin(UB, cost[idx]);

        END IF
    END IF
```

Figure 5: Pruning kernel on GPGPU

4. EXPERIMENTS AND RESULT

Both sequence and parallel version, we tested on server with CPU Intel ® Xeon ® E5-2630 2.3GHz, 24GB RAM and NVIDIA Tesla M2090 GPU with 512 cores. We have CUDA 5.5 for parallel version and *gcc* for serial version. The Table 2 shows result of computing time on serial and parallel versions of BnB algorithms and speedup.

Input:

The number of hosts (m) and the number of virtual machines (n) are entered from the keyboard.

$T_{ij}(\mathrm{i} = 1 \rightarrow \mathrm{m}, \mathrm{j} = 1 \rightarrow \mathrm{n})$, is randomly in [1,100]

$MIPS_i, PE_i, mips_j, pe_j$ are randomly generated from the sample of 4 types of hosts and 3 types of virtual machines in Sec. 3.1.

Table 2: Computing time on serial and parallel versions of BnB algorithms and speedup

Size of instance (mxn)	Time on CPU – serial version(s)	Time on CPU-GPU –parallel version(s)	Speedup
20x20	186.0	5.6	33.2
40x20	1113.0	13.1	85.6
60x20	3540.3	27.9	131.1
80x20	4700.7	36.3	130.5
100x20	8640.2	64.3	135.0

For each problem, the displayed results correspond to an average of five instances. We have observed that the best number of threads per block is 256 and number of blocks of each loop is depended on GPU memory and number of nodes in CPU pool. The proposed parallel branch and bound algorithm permits one to reduce drastically the processing time at least 30 times. The speedup depends on the size of hosts and VMs. In the table 1, the speedup will increase if the number of hosts increases because one node always generates m child nodes (see Figure 3). We note that if we increase the number of VMs, the speedup will dramatically increase but the depth of algorithm will increase so we just tested on two versions on 20 virtual machines as the depth of algorithm equal 20. The pruning kernel is very important because the best upper bound UB is constantly updated then we will prune many non-promising nodes at the same time.

We have noticed that both two versions are used Best-fit and estimate lower bound heuristic.

5. CONCLUSION AND FUTURE WORKS

In summary, we proposed here a GPU-accelerated branch-and-bound (BnB) for scheduling of virtual machines in cloud datacenters. The GPU-accelerated BnB program is faster than the serial BnB program.

At currently, we are improving the parallel BnB algorithm by keeping a number of child nodes after pruning process in GPU to forward to next iteration in GPU, instead of transfer out to host memory that means time-consuming. The forward heuristic can be feasible to apply to the application to reduce computing time.

In future work, we consider each virtual machine required in a fixed start time and duration without pre-emption in the scheduling problem. Then we will propose heuristics on searching and exploring nodes on the GPU-accelerated BnB.

REFERENCE

Chakroun, I., Melab, N., Mezmaz, M., and Tuyttens, D., Combining multi-core and GPU computing for solving combinatorial optimization problems. *Journal of Parallel and Distributed Computing* Vol. 73, Issue 12, 2013, pp. 1563–1577.

Lalami, M.E. and El-Baz, D., GPU Implementation of the Branch and Bound Method for Knapsack Problems. *Proc. 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012, pp. 1769–1777.

Spampinato, D.G. and Elstery, A.C., Linear optimization on modern GPUs, Proc. *2009 IEEE International Symposium on Parallel & Distributed Processing*, IEEE, 2009, pp. 1–8.

Quang-Hung, N., Thoai, N., and Son, N. T., EPOBF: Energy Efficient Allocation of Virtual Machines in High Performance Computing Cloud, *J. Sci. Technol. Vietnamese Acad. Sci. Technol., vol. 51, No. 4B, no. Special on International Conference on Advanced Computing and Applications (ACOMP2013)*, pp. 173–182, Oct. 2013.