Thread Scheduling for Maximizing Throughput in Multi_core systems - A Dynamic Programming Formulation

--Bharadwaj Veeravalli, Dept of ECE, NUS

Problem Scenario:

Consider a multi-core system with k Cores where each core can support 2 threads. This means that there can be at most 2k threads that can be launched to execute programs. Let us suppose that at any point in time, the system can support N memory-intensive processes to run. A process can have anywhere from one thread to many threads. When a process starts, it receives an assignment of memory and other computing resources. Whenever a thread is allocated to a process it can complete executing a certain number of instructions depending on the type of instructions. That is, different threads take different amount of time and to execute the instructions.

A program run-time trace usually determines the distribution of threads to the processes. Table 1 shows such a distribution in which allocation of thread k (row k) shows the number of instructions that can be completed by a process. It may be noted that as we increase the number of threads to a process, the number of instructions remain monotonically non-decreasing, which is expected. When memory-intensive operations are in place, threads stall and may also need to wait for other results to be available.

Given such a thread distribution, our problem is to allocate the number of threads to the processes so that the total throughput of the system is maximized, where throughput refers to the number of instructions that can be completed by the system. Thus, we are concerned with maximizing the system throughput.

Problem Formulation:

Let:

- Num of cores: 3
- Num of threads per core: 2
- Maximum num of memory-intense Processes: 4

Let x_1, x_2, x_3, and x_4, denote the number of threads allocated to Processes 1,2, 3, and 4, respectively.

Let $f_i(x_i)$, i=1,2,3,4, be the number of instructions executed upon allocating x_i threads to process i.

Table 1: Threads vs Instructions Distribution

Num of threads	Instruction Distribution			
	Process 1	Process 2	Process 3	Process 4
0	0	0	0	0
1	4	2	6	2
2	6	4	8	3
3	7	6	8	4
4	7	8	8	4
5	7	9	8	4
6	7	10	8	4

Objective: *Max (* $f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4)$)

Subjected to: $x_1 + x2 + x3 + x4 \leq 6$.

Where, $x_i \geq 0$, for all i=1,2,3,4

Solution Approach:

Stage 1	Process 1						
# of Threads x1	0	1	2	3	4	5	6
f1(x1)	0	4	6	7	7	7	7

Stage 2	Process 2	f1(x1) + f2(x2)					
# of Threads x2	f2(x2)						
0	0	4	6	7	7	7	7
1	2	6	8	9	9	9	****
2	4	8	10	11	11	****	****
3	6	10	12	13	****	****	****
4	8	12	14	****	****	****	****
5	9	13	****	****	****	****	****
6	10	****	****	****	****	****	****

Until Stage 2 (Stages 1 + 2 combined), the Optimal Solution would be:

# of Threads	Max of [f1(x1) + f2(x2)]	Thread Distrib.
0	0	0,0
1	4	1,0
2	6	2,0 or 1,1
3	8	2,1 or 1,2
4	10	2,2 or 1,3
5	12	2,3 or 1,4
6	14	2,4

Stages 1+2+3:

Stages 1+2								
# of Threads		0	1	2	3	4	5	6
Max of [f1(x1) + f2(x2)]		0	4	6	8	10	12	14
Thread Distrib.		0,0	1,0	2,0 or 1,1	2,1 or 1,2	2,2 or 1,3	2,3 or 1,4	2,4

Stage 3	Process 3	Max [f1(x1) + f2(x2)] + f3(x3)						
# of Threads x2	f3(x3)							
0	0	0	4	6	8	10	12	14
1	6	6	10	12	14	16	18	*******
2	8	8	12	14	16	18	*******	*******
3	8	8	12	14	16	*******	*******	*******
4	8	8	12	14	*******	*******	*******	*******
5	8	8	12	*******	*******	*******	*******	*******
6	8	8	********	*******	*******	*******	*******	*******

Until Stage 3 (Stages 1 + 2 + 3 combined), the Optimal Solution would be:

# of Threads	Max [f1(x1) + f2(x2) + f3(x3)]	Thread Distribution
0	0	0 0 0
1	6	0 0 1
2	10	1 0 1
3	12	2 0 1 / 1 1 1 / 1 0 2
4	14	2 1 1 / 1 2 1 / 2 0 2 / 1 1 2
5	16	2 2 1/ 1 3 1 / 2 1 2 / 1 2 2
6	18	2 3 1 / 1 4 1 / 2 2 2 / 1 3 2

Stages 1+2+3+4:

# of Threads		0	1	2	3	4	5	6
Max [f1(x1) + f2(x2) + f3(x3)]		0	6	10	12	14	16	18
Thread Distribution		0,0,0	0 0 1	1 0 1	2 0 1 / 1 1 1 / 1 0 2	2 1 1 / 1 2 1 / 2 0 2 / 1 1 2	2 2 1/ 1 3 1 / 2 1 2 / 1 2 2	2 3 1 / 1 4 1 / 2 2 2 / 1 3 2
Stage 4	Process 4	**Max [f1(x1) + f2(x2) + f3(x3)] + f4(x4)**						
# of Threads x2	f3(x3)							
0	0	0	6	10	12	14	16	18
1	2	2	8	12	14	16	18	********
2	3	3	9	13	15	17	********	
3	4	4	10	14	16	********		
4	4	4	10	14	********			
5	4	4	10	********				
6	4	4	********					

From the above table, we can see that the maximum number of instructions that can be executed is **18** and that is via thread allocation with 2 possibilities:

Optimal Solution 1	**2 2 1 1/ 1 3 1 1 / 2 1 2 1 / 1 2 2 1**
Optimal Solution 2	**2 3 1 0 / 1 4 1 0 / 2 2 2 0 / 1 3 2 0**