Algorithmic Choice in Optimization Problems: A Performance Study

Hammad Rashid, Clara Novoa, Richard Hay, Apan Qasem
Texas State University, San Marcos, TX, USA

Solving IKP with Dynamic Programming

Classical Approach
The equation below is a DP forward recursion formula to compute $f(C)$, the total profit (i.e., total value from loading the most valuable combination) if considering $m$ items and a knapsack capacity. In this recursion, Expresses the number of items possible to load, $w_m$ in the weight of item $m$ and $v_m$ in the profit of item $m$. $f(C) = \max \left\{ f(C-1) + 0, f(C-1) + v_m \right\}$, where $m = 1, 2, ..., n$; $i = 0, 1, ..., C$ (total capacity). After applying the forward recursion step, the optimal value for the profit function is given by $f(C)$. So for $n$ items (of different weights and profits) and a total capacity $C$, the most optimal profit (max possible profit) is $f(C)$.

Morales Approach
The equation below is an alternative recursion which is more efficient than the classical algorithm equation. The recursion is given by $f(n) = \max \left\{ f(n-1), f(n-1) + p_m \right\}$ where $f(n)$ represents the number of items possible to load, $w_m$ in the weight of item $m$ and $v_m$ in the profit of item $m$. $f(n) = \max \left\{ f(n-1), f(n-1) + v_m \right\}$, where $m = 1, 2, ..., n$ and $i = 0, 1, ..., C$ (total capacity).

Parallelizing IKP for CMP

Classic Parallel Algorithm
- fine-grain decomposition
- easy implementation

Tiling for locality and parallelism
- small blocks
- poor scalability
- poor performance
- high locality

Morales Parallel Algorithm
- pipelined parallelism
- coarse-grain
- efficient implementation
- small blocks
- high efficiency

Tiling for locality and parallelism
- better theoretical speedups
- better practical speedups

Results

Experimental Framework
- Core2: 2.40 GHz Intel Core 2 Quad, 4 MB L2, shared between two cores in each socket.
- Core1: 2.33 GHz Intel Core 2 Duo, 2 MB L2, shared.
- Core2: 2.66 GHz Intel Core 2 Quad, 8 MB L2, shared.
- Core1: 2.50 GHz Intel Xeon with 8 cores, with hyperthreading.

Speedup and Scalability

- weak scaling for Morales
- strong scaling for classic

Impact of Blocking Factor

Conclusion

This paper explored the issue of algorithmic choice in the context of the IKP by comparing the performance of two parallel variants on multicore platforms. The results revealed that although a row-by-row problem decomposition does not scale well when run sequentially, it exhibits good scalability when run in parallel.

Another key finding of this study is that blocking factors have significant impact on performance of each parallel variant. Therefore, to achieve improved performance, it is necessary to select blocking factors through careful analysis. Future research will explore effects on performance of the parallel variants considering data set sizes. The data locality aspects of performance will also be examined in more detail.

For additional information please contact: Hammad Rashid, Texas State University
Apan Qasem, Ph.D. Department of Computer Science
Texas State University
Email: arashid@txstate.edu
Apan Qasem, Ph.D. Department of Computer Science
Texas State University
Email: aqasem@txstate.edu
Richard Hay, Texas State University
Ingram School of Engineering
Texas State University
Email: rhay@txstate.edu

Finding Optimal Blocking Factors

Blocking factor controls both the granularity of parallelism and the data locality among concurrent threads. We observe a clear performance trend for small block sizes, picking up as we increase the block size and then drops again when we increase the block size beyond 5G. We speculate that the poor performance for smaller block sizes is a result of poor granularity. When block sizes are 4G, concurrent threads are not assigned enough computation to offset the overhead of thread creation and synchronization. This can be seen in the figure for the 11.12 Cache misses per Core. The misses continue to decrease from lower to higher block sizes and become somewhat constant after 5K.

For larger block sizes, we observe that the performance improves with larger block sizes because they provide better locality because consecutive blocks are loaded into the cache.

In this study, we have analyzed the impact of algorithmic choice on the performance of an important class of optimization problems, namely the integer knapsack problem. This paper presents a new parallel framework for solving this problem, which is based on algorithmic choice and includes a set of blocking factors. The framework is tested on a range of platforms, and the results show that it is effective in improving performance.

The Integer Knapsack Problem (IKP)

Problem
Given a knapsack of capacity $C$ and a set of $n$ different items each of them with profit $p_j$ and weight $w_j$. Find non-negative integers $x_1, ..., x_n$, where $\sum x_j \leq C$, to maximize the total profit $\sum p_j x_j$. This problem is NP-hard and is widely used in real applications in industrial engineering, e.g., budgeting, cargo loading, vehicle routing.

Importance
- widely used in real applications in industrial engineering
- directly models many practical situations such as capital budgeting, cargo loading, vehicle routing
- appears as sub-problems in many scenarios including cutting stock, set partitioning

Algorithmic Choice
- several known algorithms, e.g., dynamic programming, branch-and-bound
- algorithms amenable to different types of parallelism, e.g., data vs. pipeline vs. task parallelism
- performance highly sensitive to input data set size and shape

The rising STAR of Texas

The Integer Knapsack Problem (IKP)