Distributed Stencil Computations

Assignment 1, 5DV050, Spring 2013

Due on 7 May 2013 at 16:40

1 Purpose and aims

Theoretical analyses of performance and scalability can provide valuable insights into an algorithm's behavior on large inputs. On smaller inputs, however, the assumptions made in the theoretical modeling are no longer valid and practical experience may lead to different conclusions. The purpose of this assignment is to practice theoretical and experimental performance modeling and scalability analysis. We do this in the context of an abstract stencil computation since it is easy to understand, implement, and analyze.

2 Introduction

Consider a \( w \)-by-\( h \) regular grid as illustrated in Figure 1. The grid consists of \( w \cdot h \) grid points of which \( (w - 2) \cdot (h - 2) \) are internal (4 neighbors) and the remaining points are boundary points (< 4 neighbors). We associate with each point \((x, y)\) a real number \( u_{x,y} \in \mathbb{R} \). Each point \((x, y)\) is given an initial value and each boundary point is prescribed a fixed value via a boundary condition.

The values of the internal points evolve by the repeated application of a stencil. A stencil is simply a function that maps the old value of a point to a new value based on the values of the neighboring points. More specifically, for any internal point \((x, y)\), the stencil function \( f : \mathbb{R}^5 \to \mathbb{R} \) takes five arguments (the old value of the point and its four neighbors) and maps it into a new value for the point. As a concrete example, consider the stencil function

\[
f(u_{x,y}, u_{x-1,y}, u_{x+1,y}, u_{x,y-1}, u_{x,y+1}) = \frac{u_{x-1,y} + u_{x+1,y} + u_{x,y-1} + u_{x,y+1}}{4},
\]

which simply ignores the old value of the point \( u_{x,y} \) and replaces it with the average of the four neighboring points.

In a stencil computation, the grid evolves over many steps by the application of a stencil function to each internal point. The internal points are conceptually updated simultaneously. The inherent locality embodied in the stencil function can and should be exploited by parallel formulations. Let us therefore partition the grid into horizontal slices or blocks.
Figure 1: Illustration of a $w \times h$ regular grid. The total number of grid points is $w \cdot h$, the number of internal points is $(w - 2) \cdot (h - 2)$ and the number of boundary points is $2w + 2h - 4$.

of equal height. That is, we partition the grid into $p$ disjoint sub-grids, each $w$ points wide and approximately $h/p$ points high, as illustrated in Figure 2 for $p = 2$. Note that the cut edges (dashed in the figure) imply communication between processors since the application of a stencil function to an internal point that lies on the boundary of a process sub-grid requires access to data stored on the processor on the other side of the boundary. Note that the volume of communication is proportional to the width of the grid and completely independent of the height of the grid and the height of each partition.

Figure 2: Illustration of a $w \times h$ regular grid partitioned into two horizontal slices of height $h_0 = h_1 = 3$. A slice is stored locally on one and only one processor.

Intuitively, the ratio of communication to computation increases as the slices become thinner. For thin enough slices, the communication overhead will be significant and have a measurable impact on the parallel efficiency.
3 Theoretical analysis task

Assume that all \( p \) slices have the same height \( h/p \). Derive closed form expressions for the following functions:

- The problem size \( W(w, h) \).
- The serial runtime \( T_S(w, h) \).
- The parallel runtime \( T_P(w, h, p) \).
- The parallel overhead \( T_O(w, h, p) \).
- The parallel speedup \( S_P(w, h, p) \).
- The parallel efficiency \( E_P(w, h, p) \).

4 Performance modeling task

The aim of this task is to use the models derived in Section 3 together with estimates of the machine parameters to make predictions regarding how the yet-to-be-written program will perform.

Estimate the machine parameters introduced in Section 3 on Abisko at HPC2N based on measurements. To estimate the communication parameters, use a ping-pong benchmark. To estimate the local computation rate, measure the local computation rate of the program developed in Section 5.

Assuming that the grid is square, i.e., \( w = h \), express the size of the grid as a function of the parallel efficiency and the number of processors. How large must the grid be to obtain a 50% efficiency when \( p = 10 \)?

Assuming that the grid is \( k \) times wider than it is tall, i.e., \( w = kh \), generalize the function from the previous paragraph. How many more points must the grid contain to obtain a 50% efficiency when \( k = 10 \) compared to when \( k = 1 \), again when \( p = 10 \)?

5 Programming task

Write a parallel program in C using MPI that performs a stencil computation using the stencil function defined above. Clamp the boundary points to zero and set the values of all internal points to one. Make it possible to choose the width and height of the grid as well as the number of iterations from the command line.

A number of requirements are imposed on the implementation in order to increase the likelihood of obtaining a correct and efficient implementation that produces meaningful measurements.

- The program must be written in C using MPI. This to make my life easier.
• The program must be compiled with the PathScale compiler with optimization turned on (`-O3 -march=bdver1`). This to ensure that the generated machine code is efficient.

• Each value must be represented by a double in C. This to make communication more expensive.

• The values for each sub-grid must be stored contiguously in memory using a row-oriented memory layout. This to ensure contiguous communication buffers.

• The points should be updated in a row-oriented fashion in order to exploit spatial locality of reference within a row in the row-oriented memory layout. This to enable effective use of the cache and make the local computation faster.

• The program must use double buffering of the grid, that is, each process should have two buffers for its local slice. When the stencil function is applied, the old values should be read from one buffer and the new values should be written to the other buffer. At the end of each iteration, the roles of the two buffers should be swapped and no copy of one buffer to another should take place. This to ensure that the memory requirements stay bounded and that the cache hierarchy can be used effectively.

• Ghost points should be employed to store the boundary points as well as the points on the opposite sides of each process boundary. This to ensure that the local computation code does not need any special cases near the boundary that could slow it down.

• The iteration should be separated into a communication phase and a local computation phase. This to ensure that the communication overhead is not partially hidden by the cost of local computations.

• Each process should measure and locally store the (local) time for each phase in each iteration. Before the program halts, the average parallel cost of each phase in each iteration should be printed to standard output with one iteration per line and two columns: (1) Cost of the communication phase, and (2) cost of the local computation phase. This to ensure that you gather enough measurements.

• If indicated by an option given on the command line, the final state of the grid should be gathered on one process and printed to standard output with three digits in the fraction. This to enable a poor man’s “verification” of the correctness of the parallel implementation.

6 Experimental scalability analysis task

Before you start with this task, make sure that your program can generate reliable time measurements. In particular, consider performing a number of iterations before starting the clock to warm up the MPI library and the processors.
The aim of this task is to perform an experimental scalability analysis. The idea is to sample the parallel efficiency for a grid of problem sizes and processor counts and then use interpolation to construct an approximation to the iso-efficiency function for a given efficiency. We will compare the scalability on two types of grids: Square grids \((k = 1)\) and short-and-wide grids \((k = 10)\). Gather the necessary data as follows:

1. Based on the results of the performance modeling in Section 4, find a range of problem sizes that is suitable for both types of grids. A suitable range for the problem size is one that yields a wide spectrum of parallel efficiencies. Since we are about to search the data for a constant efficiency, we don’t want the efficiency to be more or less constant across all of our samples!

2. Allocate one node on Abisko exclusively -N 1-1 --exclusive and sample (at least) for each \(p \in \{4, 8, \ldots, 48\}\). Sample the chosen range of problem sizes at regular intervals with at least ten samples to obtain sufficiently fine resolution. More intervals yield better results.

Using the data gathered above, estimate the iso-efficiency function for a suitable choice of fixed efficiency. Note: the iso-efficiency functions are closely related to the contours of the surface \(E(p, W)\). Consider using the contour function in MATLAB.