Improving Linear Algebra Computation on NUMA platforms through auto-tuned nested parallelism

Javier Cuenca, Luis P. García, Domingo Giménez

Parallel Computing Group
University of Murcia, SPAIN
Introduction

- Scientific and engineering problems are solved with large parallel systems
- In some cases those systems are NUMA
  - A large number of cores
  - Share a hierarchically organized memory
- Kernel of the computation for those problems: BLAS o similar
  - Efficient use of routines → a faster solution of a large range of scientific problems
- Normally: multithreaded BLAS library optimized for the system is used, but:
  - If the number of cores increases → the degradation in the performance grows
- In this work:
  - An Analysis of the behaviour in NUMA of the matrix multiplication of the BLAS
  - Its combination with OpenMP to obtain nested parallelism.
  - An auto-tuning method → a reduction in the execution time
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Computational systems

- **Ben**
  - Part of the system Ben-Arab’ı of the Supercomputing Center of Murcia.
  - A shared-memory system with 128 cores.
  - HP Integrity Superdome with architecture NUMA
  - Hierarchical composition with crossbar interconnection.
  - Its architecture has two basic components:
    - the computing nodes
    - two backplane crossbars.
  - Each computing node:
    - an SMP with four CPUs dual core Itanium-2
    - an ASIC controller to connect the CPUs with the local memory and the crossbar commuters
  - The maximum memory bandwidth in a node is 17.1 GB/s and with the crossbar commuters 34.5 GB/s.
  - Access to the memory is non uniform: Four different costs in the access to the shared-memory.
  - User does not control where the threads are assigned

- **Pirineus**
  - A system at the Centre de Supercomputacio de Catalunya.
  - An SGI Altix UV 1000
    - a total of 224 Intel Xeon six-core serie 7500 (1344 cores)
    - An interconnection NUMAlink 5 in a paired node 2D torus.
  - The access to the memory is non-uniform
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
The software

- The matrix multiplication routine used: the double precision routine `dgemm`.
- The BLAS implementation of the Intel MKL toolkit used is the version 10.2
- The libraries are multithreaded: calling the routine with the desired number of threads:
- If dynamic parallelism is enabled → the number of threads is decided by the system (less than or equal to that established).
- The C compiler used was Intel icc version 11.1 in both platforms.
- Two-level parallelism: a number of OpenMP threads + calls to the multithreaded BLAS
- Matrices A and B can be multiplied with two-level parallelism:
  - $q$ threads OpenMP
  - each thread multiplying a block of adjacent rows of matrix A by the matrix B
  - establishing a number of threads ($p$) to be used in the matrix multiplication in each OpenMP thread
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Motivation

- Using a multithreaded version of BLAS \( \rightarrow \) the \texttt{dgemm} MKL routine
- The optimum numbers of threads changes from one platform to another
  - \( \rightarrow \) running restrictions and characteristics of the systems
- Default option (a number of threads equal to that of the available cores) is not good
Motivation

- **Dynamic Selection of threads:**
  - Reduction in the speed-up increases with the number of OpenMP threads
  - Number of MKL threads used is just one

- **No Dynamic Selection of threads:**
  - bigger speed-ups are obtained
  - Number of OpenMP threads grows \(\rightarrow\) an increase of the speed-up until a maximum
  - So, a large number of cores \(\rightarrow\) a good option to use a high number of OpenMP threads
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Automatic optimisation method

- Automatic Tuning System (ATS) focused on modelling the execution time
  \[ T_{\text{exe}} = f(n, SP, AP) \]
  - \( n \): the problem size
  - \( SP \): System Parameters. Characteristics of the platform (hardware + basic installed libraries)
  - \( AP \): Algorithmic Parameters. Values chosen by the ATS to reduce the execution time

- An adaptation to large NUMA platforms:
  - Each arithmetic operation: data access time depends on the relative position in the shared memory space
    - Data can be in the closest memory of the processor or in that of another processor
    - The interconnection network could be non homogeneous
  - \( \rightarrow \) those data could be at different distances from the processor that needs them
  - \( \rightarrow \) the access time is modelled with a hierarchical vision of the memory
  - It is also necessary to take into account the migration system of the platform
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Automatic optimisation method
Design phase: modelling the execution time of the routine
Modelling 1-Level: MKL multithreading \texttt{dgemm} \textit{without} generating OpenMP threads

- Model:

\[ T_{\text{dgemm}} = \frac{2n^3}{p}k_{\text{dgemm}} \]

- \textit{AP:} \( p \rightarrow \) Number of threads inside the MKL routine \texttt{dgemm}

- \textit{SP:} \( k_{\text{dgemm}} \rightarrow \) time to carry out a basic operation inside the MKL routine \texttt{dgemm} (including memory accesses)

\[ k_{\text{dgemm}} = \alpha k_{\text{dgemm\_NUMA}}(p) + (1 - \alpha)k_{\text{dgemm\_M1}} \]

- \( k_{\text{dgemm\_M1}} \): when data are in the closest memory to the operating core
- \( k_{\text{dgemm\_NUMA}} \): when data are in any level of the RAM memory
- \( \alpha \):
  - directly proportional to the use by each thread of data assigned to the other \((p-1)\) threads
  - inversely proportional to the reuse degree of data carried out by \texttt{dgemm}

\[
\alpha = \min \left\{1, \frac{p(p-1)}{\frac{n^3}{n^2}}\right\}
\]
Automatic optimisation method

Design phase: modelling the execution time of the routine
Modelling 1-Level: MKL multithreading \texttt{dgemm} without generating OpenMP threads

- Platform:
  - \(L\) memory levels
  - \(c_i\) cores have a similar access speed to the level \(l\), with \(1 \leq l \leq L\)

- \(k_{\text{dgemm\_NUMA}}\) value can be modelled, depending on \(p\):
  - If \(0 < p \leq c_1\):
    \[
    k_{\text{dgemm\_NUMA}}(p) = k_{\text{dgemm\_M1}}
    \]
  - Else if \(c_1 < p \leq c_2\):
    \[
    k_{\text{dgemm\_NUMA}}(p) = \frac{c_1 k_{\text{dgemm\_M1}} + (p - c_1) k_{\text{dgemm\_M2}}}{p}
    \]
  - …, in general, if \(c_{L-1} < p \leq c_L\):
    \[
    k_{\text{dgemm\_NUMA}}(p) = \frac{\sum_{l=0}^{L-2} (c_l - c_{l-1}) k_{\text{dgemm\_M1}} + (p - c_{L-1}) k_{\text{dgemm\_M_L}}}{p}
    \]
Automatic optimisation method

Design phase: modelling the execution time of the routine

Modelling 2-Level: OpenMP threads + MKL multithreading \( \text{dgemm} \)

- Model:
  \[
  T_{2L_{-dgemm}} = \frac{2^n}{q} \frac{nn}{p} k_{2L_{-dgemm}} = \frac{2n^3}{p} k_{2L_{-dgemm}}
  \]

- \( \text{AP:} \ p \to \) Number of threads inside the MKL routine \( \text{dgemm} \)
- \( \text{SP:} \ k_{2L_{-dgemm}} \to \) time to carry out a basic operation inside the MKL routine \( \text{dgemm} \) (including memory accesses) with \( R=pxq \) threads interactuating
  \[
  k_{2L_{-dgemm}} = \alpha k_{2L_{-dgemm\_NUMA}}(R, p) + (1-\alpha)k_{2L_{-dgemm\_M1}}
  \]

- \( k_{2L_{-dgemm\_M1}} \): when data are in the closest memory to the operating core
- \( k_{2L_{-dgemm\_NUMA}} \): when data are in any level of the RAM memory
  \[
  k_{2L_{-dgemm\_NUMA}}(R, p) = \frac{k_{\text{dgemm\_NUMA}}(R) + k_{\text{dgemm\_NUMA}}(p)}{2}
  \]

- \( \alpha \):
  \[
  \alpha = \min \left\{ 1, \frac{R(R-1)}{n^3/n^2} \right\}
  \]
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Automatic optimisation method

Installation phase: experimental estimation of the $SP$ values

- General process: calculating the $SP$ values that appear in the theoretical model

- $SP$ values to calculate: $k_{dgemm_M1}$, …, $k_{dgemm_ML}$

- Steps. For each memory level $l$, $0 \leq l \leq L$:
  1. Executing $dgemm$:
     - for a fixed (preferably small) problem size, $n$
     - for a number of threads, $p_l$, with $c_{l-1} < p_l \leq c_l$
  2. This execution time + routine model $\rightarrow k_{dgemm_{NUMA}}$ value for $p_l$ threads
  3. $k_{dgemm_{NUMA}}$ value for $p_l$ + $k_{dgemm_{NUMA}}$ model + values of $k_{dgemm_M1}$, …, $k_{dgemm_Ml-1}$ $\rightarrow k_{dgemm_Ml}$
Automatic optimisation method
Installation phase: experimental estimation of the SP values
Comparison execution vs. modelled time in platform Ben
Automatic optimisation method
Installation phase: experimental estimation of the SP values
Comparison execution vs. modelled time in platform Pirineus
Outline

- Introduction
- Computational systems
- The software
- Motivation
- **Automatic optimisation method**
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Automatic optimisation method
Execution phase: Selection of the $AP$ values

- To solve a problem with size $n$ in a concrete platform:
- The ATS takes the model of the routine, the $SP$ values calculated for this platform and the value $n$, and selects directly the most appropriate values for the $AP$ (number of OpenMP threads, $q$, and MKL threads, $p$)

<table>
<thead>
<tr>
<th>size</th>
<th>SEQ</th>
<th>MIN-MKL</th>
<th>MC-MKL</th>
<th>AUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ben</td>
</tr>
<tr>
<td>1000</td>
<td>0.320</td>
<td>0.024</td>
<td>0.091</td>
<td>0.012 (2×8)</td>
</tr>
<tr>
<td>2000</td>
<td>2.60</td>
<td>0.12</td>
<td>0.39</td>
<td>0.07 (4×16)</td>
</tr>
<tr>
<td>3000</td>
<td>8.60</td>
<td>0.32</td>
<td>0.82</td>
<td>0.23 (4×16)</td>
</tr>
<tr>
<td>4000</td>
<td>20.22</td>
<td>0.59</td>
<td>1.40</td>
<td>0.74 (4×32)</td>
</tr>
<tr>
<td>5000</td>
<td>40.23</td>
<td>1.12</td>
<td>2.11</td>
<td>1.44 (4×32)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pirineus</td>
</tr>
<tr>
<td>1000</td>
<td>0.224</td>
<td>0.034</td>
<td>0.441</td>
<td>0.021 (16×4)</td>
</tr>
<tr>
<td>2000</td>
<td>1.74</td>
<td>0.48</td>
<td>1.19</td>
<td>0.25 (8×8)</td>
</tr>
<tr>
<td>3000</td>
<td>5.46</td>
<td>0.39</td>
<td>1.31</td>
<td>0.39 (8×8)</td>
</tr>
<tr>
<td>4000</td>
<td>13.14</td>
<td>0.54</td>
<td>1.89</td>
<td>0.95 (8×8)</td>
</tr>
<tr>
<td>5000</td>
<td>25.12</td>
<td>1.13</td>
<td>2.65</td>
<td>1.02 (8×16)</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines
Conclusions and future work lines

- Behaviour of MKL matrix multiplication analysed in 2 NUMA platforms
- Number of threads equal to number of cores: Not always the best option
- Big problems in Large Systems → Use of two-level parallelism
  OpenMP+MKL is a good option
- So, a reduction in the execution time of scientific codes
  - intensively use matrix multiplications or linear algebra routines based on them
  - adequately selecting the threads to be used in the solution of the problem
- This selection: performed automatically by the auto-tuning system
  - Using a model of the execution time of each routine for each platform.
- Future:
  - Same methodology applied to other routines in linear algebra libraries
  - Different numbers and combinations of threads in different parts of the program
  - Development of multi-fabric libraries: routines run differently, depending on the problem