Linearizing Computing the Power Set with OpenMP

11th IEEE Workshop Parallel / Distributed Combinatorics and Optimization
May 17, 2021
Roger L Goodwin
Introduction

• This paper presents 4 methods for computing the power set.
  • Three methods are serial.
  • One method is parallel.

• The three serial methods are:
  • Recursive algorithm.
  • Count-in-binary algorithm.
  • Disjunctive normal form algorithms --- serial version, no OpenMP directives.

• The one parallel method is:
  • Disjunctive normal form algorithms --- parallel version, uses OpenMP directives.
Methods - Overview

• Each method will be briefly explained next.
The recursive method contains two algorithms:

1) One algorithm partitions the problem by incrementing two counters $k$ and $m$ in recursive calls.
2) The other algorithm creates the actual sets in the power set $P(\prod)$ from a set of integers.

When $m > n = |\prod|$, then the first algorithm terminates.

Both algorithms create $2^n-1$ sets in the power set

Excluding the empty set
The count-in-binary (CIB) method creates $2^n-1$ sets in the power set.

The method is simplistic:

1) Count from 0 to $2^n-1$ in decimal
2) Convert the count to binary
3) Include/exclude elements from the given set Π using the binary number
   1) 0 = exclude an element from Π
   2) 1 = include an element from Π

This particular implementation of the CIB algorithm reverses the ordering of the elements in the sets.
Methods – The Count-in-Binary Algorithm

• For example, instead of computing the set \{a, b, c\}; the CIB algorithm computes \{c, b, a\}.
  • Which are equivalent sets

• We make no effort to pad the binary numbers to the left with zeros.

• We make no effort to sort the computed sets so that the sets are more aesthetically pleasing.

• Hence, we present a Laissez-Faire CIB algorithm.
Methods – The Disjunctive Normal Form Algorithm

• The disjunctive normal form (DNF) method contains preprocessing steps.
  • The partition of the binomial coefficient function (BCF)
  • Load balancing using the partition of the BCF (PBCF)

• The BCF is as follow:

\[
\binom{n}{m} = \frac{n!}{(n-m)!m!}, \quad 0 \leq m \leq n.
\]

• The BCF is a symmetric function.
  • It sums to \(2^n\) for \(m = \{0, 1, ..., n\}\).
  • The maximum is at \(\lfloor n/2 \rfloor\) for even \(n\).
  • See graph on next slide (blue dashed line).
Methods – The Disjunctive Normal Form Algorithm
Methods – The Disjunctive Normal Form Algorithm

• We partition the BCF “given” the first element in the power set.
  • This will be explained later.

• Note: Ignore sets with cardinality 1 or $n$.

• Instead of programming the BCF function, we program the PBCF.

$$h(n, m|S = s) = \frac{\prod_{i=1}^{m-1} (n - m) - (s - 1) + i}{\prod_{i=1}^{m-1} i}$$
Methods – The Disjunctive Normal Form Algorithm

• The variable $s$ is the cardinal number to the first element in the set.

• Two noticeable characteristics of the PBCF:
  • One-half of the values are zero
  • The function drops significantly for small $s$

• Only certain arrangements in the powerset are allowed using the DNF algorithms.
  • Due to problem restrictions.

• The next slide gives an example.
Methods – The Disjunctive Normal Form Algorithm

• Suppose $\mathcal{X} = \{a, b, c, \ldots, z\}$ and $m = 2$.

• Consider the sets $\{a, b\}, \{a, c\}, \{a, z\}, \ldots, \{x, y\}, \{x, z\}, \{y, z\}$.
  • The number of sets beginning with the element $x$ is $h(n, m|S = 24) = 2$.
  • The number of sets beginning with the element $y$ is $h(n, m|S = 25) = 1$.
  • The number of sets beginning with the element $z$ is $h(n, m|S = 26) = 0$.

• But,
  • The number of sets beginning with the element $a$ is $h(n, m|S = 1) = 25$.
  • The number of sets beginning with the element $b$ is $h(n, m|S = 2) = 24$.
  • The number of sets beginning with the element $c$ is $h(n, m|S = 3) = 23$. 
Methods – The Disjunctive Normal Form Algorithm

• Remove the restriction $m = 2$. Then, $2 \leq m \leq 25$.

• Only one additional set $\{x, y, z\}$ is added to $h(n, m|S = 24)$.

• Many sets are added to $h(n, m|S = 1)$, $(n, m|S = 2)$, and $h(n, m|S = 3)$.
  • $h(n, m|S = 1) = 33,554,430$
  • $h(n, m|S = 2) = 16,777,215$
  • $h(n, m|S = 3) = 8,388,607$

• This is due to the nature of the problem.

• With the cardinality and the partition, it is possible to break-up the BCF which leads to computing the power set faster in a parallel computing environment.
Methods – The Disjunctive Normal Form Algorithm

• Next, we discuss load balancing.

• We perform load balancing before the DNF algorithms are run.

• The load balancing algorithm is a 2-pass algorithm:
  • Pass 1 computes an \((n-1) \times (n-1)\) table using the PBCF. The columns represent the cardinality \(m\). The rows represent \(s\).

  • Pass 2 redistributes the \(q\)-maximums to the rows in the table with the least number of sets (using the row totals).
Methods – The Disjunctive Normal Form Algorithm

• Load balancing ensures that a single task does not compute all of the sets with a cardinality close to \( m = \lfloor n/2 \rfloor \) and \( s \) equal to 1.

• Instead of arbitrarily setting \( q \), we calculate the \( q \)-maximums in the \((n-1) \times (n-1)\) table using the following formula:

\[
q_i = \begin{cases} 1, & \text{If } \sum_{m=1}^{n} h(n,m|S = s) < \max(i). \\ 0, & \text{Otherwise.} \end{cases}
\]

where \( i \) is bounded by \( i \in \{1, 2, ..., n-1\} \); \( \max(i) \) is the \( i \)-th largest integer in the table; the sum \( \sum_{i=1}^{n-1} q_i \) equals to the number of \( q \)-maximums.
Methods – The Disjunctive Normal Form Algorithm

• For the trivial case \( m = 1 \), the 2-pass round robin algorithm simply puts the \( n \) computations into a single task.

• The figure on the next slide compares the round robin distribution to the BCF.
  • The flat line (in black) shows the final distribution after the 2-pass round robin algorithm.
  • We prefer the flat line compared to the other two distributions when computing the power set.

• The slide after next (slide 17) shows a partial \((n-1) \times (n-1)\) table with \( n = 12 \).
  • It is always the case that the upper, center part of the table needs to be load balanced.
  • \( q = 7 \)
  • Do not forget to zero-out the \( q \)-maximums.
Methods – The Disjunctive Normal Form Algorithm
Methods – The Disjunctive Normal Form Algorithm

<table>
<thead>
<tr>
<th>m</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>Row Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>11</td>
<td>55</td>
<td>165</td>
<td><strong>330</strong></td>
<td><strong>462</strong></td>
<td><strong>462</strong></td>
<td><strong>330</strong></td>
<td>165</td>
<td>55</td>
<td>11</td>
<td>2,058</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>45</td>
<td>120</td>
<td><strong>210</strong></td>
<td><strong>252</strong></td>
<td><strong>210</strong></td>
<td>120</td>
<td>45</td>
<td>10</td>
<td>1</td>
<td>1,023</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>36</td>
<td>84</td>
<td>126</td>
<td>126</td>
<td>84</td>
<td>36</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>511</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>28</td>
<td>56</td>
<td>70</td>
<td>56</td>
<td>28</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>21</td>
<td>35</td>
<td>35</td>
<td>21</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Methods – The Disjunctive Normal Form Algorithm

• Some notes:
  • The most computer intensive sets to compute are those sets that have been redistributed by the round robin algorithm.
  
  • The round robin algorithm implements the $q$-maximums by writing snippets of code which has to be inserted into the tasks.
  
  • Using both the PBCF and the 2-pass round robin algorithm, the entire power set for $n = 26$ can be computed in 55 seconds on the laptop used in [5], [6]. This is a 42.7% reduction in run-time.
Methods

• The differences between the CIB method and the DNF method:
  • The CIB method contains two algorithms. The DNF method contains $m-1$ algorithms --- one algorithm for each cardinality $m, 1 \leq m \leq n-1$.

  • The CIB algorithm terminates after $2^n-1$ iterations. The DNF algorithms terminate after a pre-determined maximum has been reached.

  • The CIB main loop contains a single DO loop. The DNF algorithms’ loops contain $m$ loops for each cardinality.
Methods

• Advantages of each method:
  • The recursive algorithm is easy to program.
  • The CIB algorithm is easy to understand and easy to program.
  • The DNF algorithm runs in linear time in a parallel computing environment.

• Disadvantages of each method:
  • The recursive algorithm runs in exponential run-time.
  • The CIB algorithm runs in exponential run-time.
  • The DNF algorithm requires pre-processing.
Empirical Evaluation

• We evaluate the methods on the Stampede2 supercomputer using the Skylake (SKX) compute nodes.

• We will:
  • Construct a task graph to show potential parallelism.
  • Run the Intel Advisor to show the top 5 time consuming loops.
  • Construct a scalability curve.
  • Summarize the results of the algorithms.
  • Outline a method to compute the power set for large $n$.  

Empirical Evaluation

• We construct a task graph of the DNF algorithm to show the possible parallelism in the program.

• The widest part of the graph shows the possible parallelism.

• The critical path shows maximum run-time of the program.
  • Because computing 5,200,300 sets with 14 nested loops is more computer intensive than computing 5,200,300 sets with 13 nested loops.

• See the Figure on the next slide.
Empirical Evaluation
Empirical Evaluation

• The Intel Advisor is a source code profiling tool.

• The Intel Advisor shows the top 5 time consuming loops.
  • The algorithms with the largest number of sets to compute and those algorithms with the greatest number of nested loops about the center $n/2$.
  • Take the most time to compute the power set.

• See the Table on the next slide.
## Empirical Evaluation

<table>
<thead>
<tr>
<th>Loop</th>
<th>Self-Time</th>
<th>Total Time</th>
<th>Trip Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in dnf_new_14]</td>
<td>3.582 s</td>
<td>16.330 s</td>
<td>1</td>
</tr>
<tr>
<td>[loop in dnf_new_13]</td>
<td>3.511 s</td>
<td>16.360 s</td>
<td>1</td>
</tr>
<tr>
<td>[loop in dnf_new_12]</td>
<td>3.210 s</td>
<td>14.029 s</td>
<td>1</td>
</tr>
<tr>
<td>[loop in dnf_new_15]</td>
<td>2.779 s</td>
<td>13.960 s</td>
<td>1</td>
</tr>
<tr>
<td>[loop in dnf_new_11]</td>
<td>2.441 s</td>
<td>10.400 s</td>
<td>1</td>
</tr>
</tbody>
</table>
Empirical Evaluation

• The Intel Advisor Source Code Profiling tool is useful at times

• It can be used to confirm information you already know

• For instance, the top 5 time-consuming loops
  • You know which ones they are
  • The Intel Advisor tool confirms this
Empirical Evaluation

• Some notes on the Intel Advisor.

• The Intel Advisor suggests changing the data type inside the loop so that it matches
  • This will have a better chance of using the full vector register width
  • I modified the code to use the I(4) data type
  • The code ran twice as slow
Empirical Evaluation

• The next slide shows the scalability curve for the OpenMP DNF algorithm.
  • The scalability graph shows a linear relationship between cores versus speed-up.

• We estimated the percentage amount of serial code using Equation (6) in the paper.

• Then applied Amdahl’s law to obtain the scalability curve.
Empirical Evaluation
Empirical Evaluation

- The following table summarizes the results of the timing study of the different methods.

<table>
<thead>
<tr>
<th>n</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>Tp</th>
<th>q</th>
<th>CV</th>
<th>Sp</th>
<th>Ep</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.04</td>
<td>0.04</td>
<td>0.2</td>
<td>0.2802</td>
<td>9</td>
<td>0.6</td>
<td>0.7</td>
<td>1.4%</td>
</tr>
<tr>
<td>16</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1832</td>
<td>10</td>
<td>0.3</td>
<td>1.2</td>
<td>2.5%</td>
</tr>
<tr>
<td>17</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.2549</td>
<td>11</td>
<td>0.7</td>
<td>1.0</td>
<td>2.0%</td>
</tr>
<tr>
<td>18</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2155</td>
<td>12</td>
<td>0.5</td>
<td>1.3</td>
<td>2.8%</td>
</tr>
<tr>
<td>19</td>
<td>0.8</td>
<td>0.7</td>
<td>0.3</td>
<td>0.1174</td>
<td>13</td>
<td>0.2</td>
<td>2.8</td>
<td>5.9%</td>
</tr>
<tr>
<td>20</td>
<td>1.6</td>
<td>1.5</td>
<td>0.4</td>
<td>0.2557</td>
<td>14</td>
<td>0.7</td>
<td>1.7</td>
<td>3.5%</td>
</tr>
<tr>
<td>21</td>
<td>3.3</td>
<td>3.1</td>
<td>0.6</td>
<td>0.1097</td>
<td>15</td>
<td>0.1</td>
<td>5.4</td>
<td>11.3%</td>
</tr>
<tr>
<td>22</td>
<td>7.1</td>
<td>6.5</td>
<td>1.0</td>
<td>0.1770</td>
<td>16</td>
<td>1.1</td>
<td>5.5</td>
<td>11.4%</td>
</tr>
<tr>
<td>23</td>
<td>14.5</td>
<td>13.5</td>
<td>1.8</td>
<td>0.1487</td>
<td>17</td>
<td>0.8</td>
<td>11.8</td>
<td>24.6%</td>
</tr>
<tr>
<td>24</td>
<td>30.6</td>
<td>28.0</td>
<td>3.4</td>
<td>0.1149</td>
<td>18</td>
<td>0.3</td>
<td>29.3</td>
<td>61.0%</td>
</tr>
<tr>
<td>25</td>
<td>62.7</td>
<td>58.2</td>
<td>6.7</td>
<td>0.1733</td>
<td>18</td>
<td>0.4</td>
<td>38.5</td>
<td>80.3%</td>
</tr>
<tr>
<td>26</td>
<td>131.4</td>
<td>117.4</td>
<td>13.6</td>
<td>0.2746</td>
<td>20</td>
<td>0.5</td>
<td>49.5</td>
<td>103.1%</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>0.280196</td>
<td></td>
<td></td>
<td>12.4</td>
<td>25.8%</td>
</tr>
<tr>
<td>Min</td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td></td>
<td></td>
<td>0.7</td>
<td>1.4%</td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td></td>
<td>131.4</td>
<td></td>
<td></td>
<td>49.5</td>
<td>103.1%</td>
</tr>
</tbody>
</table>
Empirical Evaluation

• $T_0 =$ recursive algorithm (seconds)
• $T_1 =$ CIB algorithm (seconds)
• $T_2 =$ non-OpenMP DNF algorithms (seconds)
• $T_p =$ OpenMP DNF algorithms (seconds)

• Only one coefficient of variation (CV) above 1.0
  • Much variability about the average.

• We obtain an efficiency (EP) of 100% (accounting for variability) because the serial algorithm $T_2$ ran poorly and the parallel algorithm $T_p$ ran efficiently.
Empirical Evaluation

• The serial algorithms $T_0$, $T_1$, and $T_2$ have exponential run-time curves.

• The $T_p$ parallel algorithm (OpenMP DNF algorithm) has a non-exponential run-time curve.

• The graph on the next slide shows the graph for the input sizes versus the run-times for the OpenMP DNF algorithm.
Empirical Evaluation
Empirical Evaluation

• Recommendations from the timing study

• The Stampede2 supercomputer is a shared machine
  • If another job is thrashing while your job is running, this will affect your timing study
  • Run your job numerous times on different days to get a good timing
Configuration Management

• We computed the power set in its entirety for \( n = 15, 16, \ldots, 26 \).

• Consider computing the power set for \( n = 150 \) and \( n = 45,136 \).

• Obvious some implementation limitations will come up:
  • Integer exceeds machine limits.
  • Segmentation fault.
  • A single user can only have 25 jobs in queue at a time.
Configuration Management

• Some possible workarounds include:
  • Use the –fno-range-check option when compiling the code.
  • Overwrite the values in the array when the index reaches $2^{31}$.
  • Wait until some of the jobs have finished, then submit more jobs.

• The $q$-maximums are a second source of exponentiation.
  • These values must be partitioned into smaller sets.
  • We divide by $2^{n-15}$. This is also the required number of threads.
  • Leave the remaining distribution as-is from the 2-pass round robin algorithm.
Configuration Management

• We conduct a small timing study up to $n = 150$ as a proof of concept.

• The max time always occurs at the largest $q$-maximum $\max_{(n,1)}$ for any $n$.

• This small timing study saves a single computation from a single partition from $\max_{(n,1)}$ for $n = 15, \ldots, 150$.

• The next slide shows a graph of the input size versus the run-times.
  • Using multiple nodes and multiple cores.
Configuration Management

\[
y = 0.0008n + 0.0883
\]
\[
R^2 = 0.7848
\]
Configuration Management

• The graph on the previous slide has 2 plateaus.
  • These plateaus are probability due to the amount of nested loops as $n$ increases.

• Additional obstacles must be overcome before computing large power sets:
  • Compute the factorial of a number larger than $n = 150$; say $n = 1,000$ to $45,136$.
  • Automatically monitor the queue; and submit a batch job when one job has finished.
Configuration Management

• Using the model from the timing study $y = 0.0008n + 0.0883$, it is estimated that it will take 36.1971 seconds to compute the largest partition for the power set for $n = 45,136$.
  • The remaining tasks are smaller and will take less time.
  • On the Stampede2 SKX compute nodes
Questions

• Thank you for attending.

• Does anyone have any questions?

• Profile and research: https://rogerlgoodwin.brandyourself.com