Queue Processor Architecture for Novel Queue Computing Paradigm Based on Produced Order Scheme

Ben A. Abderazek, M. Arsenji, S. Shigeta, T. Yoshinaga, and M. Sowa

The University of Electro-Communications
Graduate School of Information Systems, Tokyo
Overview of Queue based computation Models

- Queue based computation models (QCM) are based on first-in-first-out data structure.

Queue Based Computation Models were proved to be:
- Also efficient in evaluating a given expression [1].
- Program written for different QCM models were found to be:
  - Short in term of program size [2]
  - Parallelism Efficient due to the nature parallelism of QCM

New computation models were proposed by and were found to be:
- More Efficient in term of short program size [1]
- More Efficient in term of Program Degree of parallelism [2]
In this talk we will present a novel processor Queue architecture

- The proposed system uses Queue Registers to store intermediate results.
- Datum are inserted in the Queue Register in produced order scheme → can be reused.

Queue Computation Unit calculates the operands addresses for each instruction.

Live data are protected by the LQH pointer.
Queue instructions sequence are derived from a so called level-order-Scan Tree (LOST) algorithm.

- From the deepest to the shallowest.
  - (n1, n2, n3, n4, n5, n6, n7, n8, n9)
Fig. 1. Data Flow Graph
(a) Original Sample Graph  (b) Translated sample Graph
Produced Order Computing

```plaintext
ld a  # load variable "a"
ld b  # load variable "b"
ld c  # load variable "c"
ld d  # load variable "d"
mul  # multiply the first two variables
add  # from the front of the Queue
div -2  # divide the entry pointed by QH by the variable "c"
mul -1  # which is located at a negative offset "-2" from QT
st e  # store (ab/c) into "e"
st f  # store a*b*(c+d) into "f"
```

Fig. 2. Produced order sample program  
(a) Assembly code, (b) Queue contents at each execution stage
**Queue Head and Tail Calculations**

\[ QHP \ (i+1) = QHP \ (i) + \sum (Consumed \ Data) \]

Where, *Consumed Data* is the number of fetched operands of an instruction.

The QTP calculation of an instruction \((i+1)\) is calculated by:

\[ QTP \ (i+1) = QTP \ (i) + \sum (Produced \ Data) \]

Where, *Produced Data* is the number of the results generated by an instruction.
Fig. 3. System Architecture Basic Block
System Architecture

PQP System Architecture
Instruction Set Architecture

- Variable instructions length.
- Zero-operand instructions are one byte wide and
- All single operand instructions are two bytes wide.
- Memory instructions, which are also single-operand, are three bytes wide.
- The instruction set is classified into three classes:
  - **Basic class**: consists of all arithmetic, shift, rotate, compare and logical operations.
  - **Mem class**: consists of the memory, control, branch, and loop operations.
  - **MemIndex Class**: consists of memory indexed address, control, branch and loop operations.
Evaluation (1/5)

- In this evaluation, we want to find the optimal number of FUs and the fetch width (FW), which are two major parameters of the proposed processor architecture.

- The EILP (executed instruction level parallelism), over a range of FUs and FW, is evaluated.

- We used four sample benchmark programs (FFT, Newton, Prefix, and LU-Decompositions). The FFT32 benchmark is a 32-point Fast Fourier Transformation program.

- The Newton benchmark is a 4-degree Newton polynomial interpolation program.

- The Prefix benchmark is a prefix computation method of degree 8, which treats a given equation as a system of recurrences.
**Evaluation (2/5)**

Fig. 10 EIPC for SC with OoO Issue Policy

Fig. 9 EIPC for SC with IO Issue Policy

Table 1. Benchmark programs characteristics

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Max. ILP</th>
<th>Dist. Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA (FFT32)</td>
<td>High</td>
<td>Long</td>
</tr>
<tr>
<td>SB (LU)</td>
<td>High</td>
<td>Short</td>
</tr>
<tr>
<td>SC (Prefix)</td>
<td>Low</td>
<td>Long</td>
</tr>
<tr>
<td>SD (Newton)</td>
<td>Low</td>
<td>Short</td>
</tr>
</tbody>
</table>
As a summary, since the decision of the number of FUs, which is fixed to be (9, 9, 9, 9), is made according to The used Benchmark programs characteristics, the optimal FW (in term optimal EILP) is 12 Bytes.
programs written for PQPpvc are larger than programs written for PQPpv.

In average, programs for PQP pv are 10 % to 26 % shorter than programs for PQPcv.
This is due to the storing order of data and to the way data are consumed within the OPQ
Conclusions (1)

- In this paper we proposed parallel queue processor architecture (PQPpv) based on produced order queue computation model.

- First, we found that, in average, programs for PQPpv are 10 % to 26 % shorter than programs for PQPcv.

- PQPcv programs size were also found in our earlier research 66% smaller than programs for RISC (SPARC) code [14].

- From the evaluation, we expect that programs for PQPpv are about 50% shorter than those for RISC code.
Second, to find the optimal fetch width and the Functional units number, we used several benchmark programs that exhibit specialized forms of parallelism was simulated for two different execution schemes (in program order (IO) and out of program order (OOO)).

From the above evaluation, we conclude that the performance of the PQP processor linearly increases with the increase of functional units’ number for a fixed fetch width. However, it saturates in a certain point. The optimal FUs and FW are (9, 9, 9, 9) and 15 Bytes respectively.

The above parameters almost feat for all types of configurations we have used during our evaluation.
Thank you for your Attention.