The Anytime Automaton

Joshua San Miguel
Natalie Enright Jerger
Summary

We propose the **Anytime Automaton**: 

- A new computation model for approximate computing.
We propose the **Anytime Automaton**:  
- A new computation model for approximate computing.
We propose the **Anytime Automaton**:

- A new computation model for approximate computing.
Approximate Computing

Many applications are inherently noisy and imprecise.

<table>
<thead>
<tr>
<th>Data mining</th>
<th>Computer vision</th>
<th>Audio and video processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="http://www.zentut.com/" alt="Fingerprint" /></td>
<td><img src="http://www.cc.gate.edu/~cnieto6/" alt="Image" /></td>
<td><img src="http://themusicparlour.blogspot.ca/" alt="Waveform" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gaming</th>
<th>Machine learning</th>
<th>Dynamical simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="http://www.businessweek.com/" alt="Gaming" /></td>
<td><img src="http://www.analyticbridge.com/" alt="Neural network" /></td>
<td><img src="http://www.scientific-computing.com/" alt="Simulation" /></td>
</tr>
</tbody>
</table>
Many applications are inherently noisy and imprecise.

<table>
<thead>
<tr>
<th>Data mining</th>
<th>Computer vision</th>
<th>Audio and video processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaming</td>
<td></td>
<td>Dynamical simulation</td>
</tr>
</tbody>
</table>

But how can we apply approximate computing techniques and still ensure acceptability in final output?
Approximate Computing

program ()
{
    foos_on_first();
    bars_on_second();
    hello_worlds_on_third();
}

time

foos_on_first        bars_on_second        hello_worlds_on_third
program () {
    approx_foos_on_first();
    bars_on_second();
    hello_worlds_on_third();
}

time
tune quality
(runtime-quality tradeoff)
program ()
{
    \textit{approx\_foos\_on\_first}();
    bars_on_second();
    hello_worlds_on_third();
}

\textit{foos\_on\_first} \quad \textit{bars\_on\_second} \quad \textit{hello\_worlds\_on\_third}

time

tune quality
\textit{(runtime-quality tradeoff)}
program ()
{
    \textit{approx\_foos\_on\_first}();
    \textit{approx\_bars\_on\_second}();
    hello\_worlds\_on\_third();
}

\textbf{time}
\begin{itemize}
    \item \textbf{foos\_on\_first} \\
    \item \textbf{bars\_on\_second} \\
    \item \textbf{hello\_worlds\_on\_third}
\end{itemize}
\textbf{tune quality}
program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

time

foos_on_first bars_on_second hello_worlds_on_third
tune quality
Approximate Computing

program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

But final output may not be acceptable!
Approximate Computing

Program()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

Difficult to ensure acceptability of final output on-the-fly, since quality control limited to local approximations and not their composition.

(Challenge #1: Holistic Quality Control)

But final output may not be acceptable!
Approximate Computing

Difficult to ensure acceptability of final output on-the-fly, since quality control limited to local approximations and not their composition.

(Challenge #1: Holistic Quality Control)
Real-Time Computing

Real-time systems impose strict runtime constraints; loss in output quality more tolerable than not finishing in time.

<table>
<thead>
<tr>
<th>Streaming multimedia</th>
<th>Automotive systems</th>
<th>Telecommunications</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="http://www.streamingvideoprovider.co.uk/" alt="Streaming multimedia" /></td>
<td><img src="http://www.beaudaniels-illustration.com/" alt="Automotive systems" /></td>
<td><img src="http://m.exed.hec.edu/" alt="Telecommunications" /></td>
</tr>
</tbody>
</table>
program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

foos_on_first  bars_on_second  hello_worlds_on_third

strict target runtime
Real-Time Computing

program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

foos_on_first    bars_on_second    hello_worlds_on_third

time            tune quality      tune quality      strict target runtime
Real-Time Computing

program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

foos_on_first  bars_on_second  hello_worlds_on_third

time  tune quality  tune quality  strict target runtime
Real-Time Computing

program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

But actual runtimes variable!

foos_on_first

bars_on_second

hello_worlds_on_third

strict target runtime
Real-Time Computing

Difficult to ensure strict real-time constraints are met (i.e., interrupt the application), since runtime-quality tradeoffs vary dynamically.

(Challenge #2: Interruptibility)
In user-interactive environments, users dictate quality requirements on-the-fly.
program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}
User-Interactive Computing

program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

tune quality?
program ()
{
    approx_foos_on_first();
    approx_bars_on_second();
    hello_worlds_on_third();
}

foos_on_first

bars_on_second

hello_worlds_on_third

time

tune quality?

But target quality unknown!
User-Interactive Computing

Difficult to ensure acceptability for a given user at a given context, since acceptable quality cannot be determined a priori. 

(Challenge #3: User Flexibility)
We propose the **Anytime Automaton**:

- A new computation model for approximate computing.
- Revisits and generalizes concepts from anytime (or iterative) algorithms, originally studied for real-time decision problems.
- A *recipe* for applying approximate computing techniques such that the final output is available early and improves in quality over time.
Anytime Automaton

quality

application execution
Anytime Automaton

quality

application execution

final output

conventionally, single output
Anytime Automaton

quality

application execution

precise output
Anytime Automaton

holistic quality control: final output available early
Anytime Automaton

quality

application execution

strict target runtime

precise output
Anytime Automaton

interruptibility: use current output if needed

quality

application execution

strict target runtime

precise output
Anytime Automaton

quality

application execution

precise output
Anytime Automaton

quality

precise output

application execution
Anytime Automaton

quality

application execution

precise output
Anytime Automaton

quality

application execution

precise output

user flexibility:
wait longer for better quality
Outline

Anytime Automaton
  ➢ The Model
  ➢ The Approximations

Evaluation
  ➢ Methodology
  ➢ Experimental Results

Conclusion
Anytime Automaton

program:

data dependence

computation
Anytime Automaton

program:

\[ \text{inst A;} \]
Anytime Automaton

program:

inst A;
inst B;
inst C;
Anytime Automaton

program:

```
func (...);
```
Anytime Automaton

program:

```c
kernel () {
    ...
}
```
program:
Dataflow Model
Dataflow Model
Dataflow Model
Dataflow Model
Dataflow Model
Dataflow Model
Dataflow Model
Dataflow Model
produce a single result
produce multiple approx versions of result (i.e., anytime)

precise version
Anytime Automaton

Data Diffusion Model
Anytime Automaton
Anytime Automaton

child works on parent’s approx result
Anytime Automaton

child works on parent’s approx result

parent works on producing better approx result
Anytime Automaton
Anytime Automaton
Anytime Automaton

Input

final output available early
Anytime Automaton
Anytime Automaton
Anytime Automaton
Anytime Automaton
Anytime Automaton
Anytime Automaton
Anytime Automaton – The Model

1. Ensure precise output is always produced eventually.
1. Ensure precise output is always produced eventually.
1. Ensure precise output is always produced eventually.

Anytime Automaton – The Model

```
kernel () {
  ...
}
```
1. Ensure precise output is always produced eventually.
1. Ensure precise output is always produced eventually.
2. Create the effect of improving accuracy over time.
2. Create the effect of improving accuracy over time.

Anytime (or iterative) algorithms have been studied before but are traditionally built into the *coarse-grained* derivation of an application.

Approximate computing techniques have proliferated recently and have been shown to have general *fine-grained* applicability.
2. Create the effect of improving accuracy over time.
2. Create the effect of improving accuracy over time.
2. Create the effect of improving accuracy over time.
2. Create the effect of improving accuracy over time.

approx computing techniques

kernel () {
  ...
}

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
Anytime Automaton – The Model

2. Create the effect of improving accuracy over time.
Anytime Automaton – The Model

3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
Anytime Automaton – The Model

3. Enable interruptibility via pipelining.
3. Enable interruptibility via pipelining.
Anytime Automaton – The Model

3. Enable interruptibility via pipelining.

```
computation A

computation B

computation C
```

`approx output ready!`
1. General case: apply approximations \textit{iteratively}. 
1. General case: apply approximations *iteratively*.
1. General case: apply approximations \textit{iteratively}.

- Loop perforation
- Smaller perforation stride
- Floating-point precision
- More mantissa bits
- SRAM bit upsets
- Higher supply voltage
- Load value approximation
- Lower approximation degree
- Neural acceleration
- Higher neural network complexity
1. General case: apply approximations iteratively.

- Loop perforation
1. General case: apply approximations *iteratively*.

- Loop perforation

perforation stride 20: for $i = 0, 20, 40, 60, 80, 100, \ldots, N-1$
1. General case: apply approximations iteratively.

Loop perforation

- **perforation stride 20:** for $i = 0, 20, 40, 60, 80, 100, \ldots, N-1$
- **perforation stride 15:** for $i = 0, 15, 30, 45, 60, 75, \ldots, N-1$
- **perforation stride 10:** for $i = 0, 10, 20, 30, 40, 50, \ldots, N-1$
- **perforation stride 5:** for $i = 0, 5, 10, 15, 20, 25, \ldots, N-1$
- **perforation stride 1:** for $i = 0, 1, 2, 3, 4, 5, 6, 7, \ldots, N-1$
1. General case: apply approximations \textit{iteratively}.

- Loop perforation

- Perforation stride $20$: for $i = 0, 20, 40, 60, 80, 100, \ldots, N-1$

- Perforation stride $15$: for $i = 0, 15, 30, 45, 60, 75, \ldots, N-1$

- Perforation stride $10$: for $i = 0, 10, 20, 30, 40, 50, \ldots, N-1$

- Perforation stride $5$: for $i = 0, 5, 10, 15, 20, 25, \ldots, N-1$

- Perforation stride $1$: for $i = 0, 1, 2, 3, 4, 5, 6, 7, \ldots, N-1$

Achieves desired effect of improving quality over time, but can yield redundant work.
1. General case: apply approximations iteratively.

- Loop perforation

- **perforation stride 20:** for \( i = 0, 20, 40, 60, 80, 100, \ldots, N-1 \)

- **perforation stride 15:** for \( i = 0, 15, 30, 45, 60, 75, \ldots, N-1 \)

- **perforation stride 10:** for \( i = 0, 10, 20, 30, 40, 50, \ldots, N-1 \)

- **perforation stride 5:** for \( i = 0, 5, 10, 15, 20, 25, \ldots, N-1 \)

- **perforation stride 1:** for \( i = 0, 1, 2, 3, 4, 5, 6, 7, \ldots, N-1 \)
2. Better case: apply **diffusive** approximations.

- Each approximation builds on the previous one.
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.

- Each approximation builds on the previous one.
2. Better case: apply **diffusive** approximations.

- Each approximation builds on the previous one.

![Diagram showing data dependences and progression from simple to complex approximations.]

**Data dependences**

(Each approximate result contributes usefully to precise result)

**Data sampling**

- More samples

**Integer/fixed-point precision**

- More bits
2. Better case: apply **diffusive** approximations.
   - Input sampling (e.g., generating a distribution)
2. Better case: apply **diffusive** approximations.

- Input sampling (e.g., generating a distribution)
2. Better case: apply **diffusive** approximations.

- Input sampling (e.g., generating a distribution)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Input sampling (e.g., generating a distribution)

To improve quality, no need to reiterate from beginning; therefore, **diffusive**.
(e.g., just add more samples to current result)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.

- Input sampling (e.g., generating a distribution)
2. Better case: apply **diffusive** approximations.

- Input sampling (e.g., generating a distribution)
2. Better case: apply **diffusive** approximations.

- Input sampling (e.g., generating a distribution)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Input sampling (e.g., generating a distribution)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Input sampling (e.g., generating a distribution)

Minimal redundant work since each element processed exactly once.
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.

- Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.
- Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.

- Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.

- Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.

- Output sampling (e.g., generating an image)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Output sampling (e.g., generating an image)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   - Output sampling (e.g., generating an image)

```
    tree permutation

    [Grid]
```


Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.

- Output sampling (e.g., generating an image)
2. Better case: apply *diffusive* approximations.

- Output sampling (e.g., generating an image)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.
   ➢ Output sampling (e.g., generating an image)
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
X & Y & Z
\end{bmatrix}
\bullet
\begin{bmatrix}
10.1101 & 01.0010 & 11.0110
\end{bmatrix}
\]
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
  X \\
  Y \\
  Z
\end{bmatrix} \bullet \begin{bmatrix}
  10.1101 \\
  01.0010 \\
  11.0110
\end{bmatrix}
\]

\[
\begin{array}{ccc}
  X \times 10.1101 & Y \times 01.0010 & Z \times 11.0110 \\
  \hline
  \text{time}
\end{array}
\]
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
X & Y & Z
\end{bmatrix} \cdot 
\begin{bmatrix}
10.1101 & 01.0010 & 11.0110
\end{bmatrix}
\]

\[
\begin{array}{c}
X \times 10.1101 \\
Y \times 01.0010 \\
Z \times 11.0110
\end{array}
\]

\text{time}
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)
Anytime Automaton – The Approximations

2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
X & Y & Z
\end{bmatrix} \cdot \begin{bmatrix}
10.1101 & 01.0010 & 11.0110
\end{bmatrix}
\]
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
X & Y & Z \\
\end{bmatrix} \cdot \begin{bmatrix}
10.1101 & 01.0010 & 11.0110 \\
\end{bmatrix}
\]
2. Better case: apply **diffusive** approximations.

- Integer/fixed-point precision (e.g., dot product)

\[
\begin{bmatrix}
X & Y & Z
\end{bmatrix} \cdot \begin{bmatrix}
10.1101 & 01.0010 & 11.0110
\end{bmatrix}
\]

- \(\text{approx result ready!}\)
More details in paper:

- Asynchronous/synchronous pipelining
- Data locality with sampling
- Approximate storage techniques
- Thread scheduling
Experiments:

- IBM Power 780 system
  - 4 POWER7+ cores
  - 32 total hardware threads

Applications:

- PERFECT and AxBench suites
  - 2D convolution (output sampling, reduced precision†, SRAM bit upsets†)
  - debayer (output sampling)
  - discrete wavelet transform (loop perforation)
  - histogram equalization (input and output sampling)
  - k-means clustering (output sampling)

†see paper
Evaluation – 2D Convolution

![Graph showing SNR (dB) vs. runtime (normalized to baseline)]
Evaluation – 2D Convolution

SNR (dB) vs. runtime (normalized to baseline). The graph shows a comparison between different SNR levels and their corresponding runtimes. The data points indicate that as the SNR increases, the runtime decreases, suggesting a better performance in terms of efficiency.
Evaluation – 2D Convolution

SNR (dB) vs. runtime (normalized to baseline)

SNR (dB)
0 10 20 30 40
runtime (normalized to baseline)
0 0.5 1 1.5 2
inf
better

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
Evaluation – 2D Convolution

![Bar Chart]

- SNR (dB)
- Run time (normalized to baseline)
- Better

inf
Evaluation – 2D Convolution

![Graph showing SNR vs runtime (normalized to baseline)](image)

SNR (dB) vs runtime (normalized to baseline) for 2D Convolution evaluation. The graph indicates a trade-off between SNR and runtime, with better performance at lower runtime values.
Evaluation – 2D Convolution

![Chart showing SNR (dB) vs. runtime (normalized to baseline)]
Evaluation – 2D Convolution

![Graph showing SNR vs. runtime for 2D Convolution. The x-axis represents runtime normalized to baseline, ranging from 0 to 2. The y-axis represents SNR in dB, ranging from 0 to 40. The graph includes data points indicating better performance as SNR increases with increasing runtime.](image-url)
Evaluation – Debayer

SNR (dB) vs runtime (normalized to baseline)

better
Evaluation – Debayer

![SNR vs Runtime Plot]

- SNR (dB) values range from 0 to 20.
- Runtime (normalized to baseline) values range from 0 to 1.8.
- The plot shows data points indicating performance improvement.

Better performance is indicated by higher SNR values.
Evaluation – Debayer
Evaluation – Debayer

![Graph showing SNR (dB) vs. runtime (normalized to baseline)]
Evaluation – Debayer

SNR (dB) vs. runtime (normalized to baseline)

- Better results are indicated by a lower runtime and higher SNR.
- The graph shows a comparison of different SNR levels with corresponding runtimes.

- The orange diamonds represent the data points for the Debayer algorithm.
- The green dashed line at the right end of the x-axis indicates infinite runtime.

The graph illustrates the performance of the Debayer algorithm across various SNR levels and normalized runtimes.
Evaluation – Discrete Wavelet Transform

![Graph showing SNR (dB) vs. runtime (normalized to baseline). The x-axis represents runtime normalized to baseline, ranging from 0 to 2.5. The y-axis represents SNR in dB, ranging from 0 to 30. The graph includes a legend indicating 'better' with an upward arrow pointing to the right.]
Evaluation – Discrete Wavelet Transform

![Graph showing SNR (dB) vs. runtime (normalized to baseline)](image)

- Better results are indicated by a higher SNR (decibels) value and a lower runtime (normalized to baseline).

- The graph shows a trend where increasing SNR corresponds to improved performance, as indicated by the upward arrow.
Evaluation – Discrete Wavelet Transform

![Graph showing SNR (dB) vs. runtime (normalized to baseline)]

- SNR (dB) values range from 0 to 30.
- Runtime is normalized to baseline.
- The graph indicates better performance as SNR increases.

Legend:
- Better performance is represented by the upward arrow.
Evaluation – Discrete Wavelet Transform

SNR (dB) vs runtime (normalized to baseline)

SNR (dB)

better

inf

Runtime (normalized to baseline)
Evaluation – Discrete Wavelet Transform

SNR (dB) vs. runtime (normalized to baseline)

- Better performance indicated by higher SNR values.
- The graph shows a trend where higher SNR values correspond to better performance.

Inf indicates the upper limit of the runtime.

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
how acceptable the output is

application execution

how much time is expended
Conclusion

We propose the **Anytime Automaton**:  
➢ A new computation model for approximate computing.
We propose the **Anytime Automaton**:  

- A new computation model for approximate computing.

**holistic quality control:**  
final output available early

**precise output**
Conclusion

We propose the **Anytime Automaton**:  
- A new computation model for approximate computing.

![Diagram showing the Anytime Automaton concept]

**Quality** vs. **Application Execution**

- **Interruption:** Use current output if needed.
- **Precise Output:**
We propose the **Anytime Automaton**: 
- A new computation model for approximate computing.

User flexibility:
- Wait longer for better quality

Quality vs. application execution chart.
Thank you

The Anytime Automaton

Joshua San Miguel
Natalie Enright Jerger

Special thanks to IBM collaborators:
Viji Srinivasan, Ravi Nair, Dan Prener