Performance Optimization of Multi-Core Grammatical Evolution Generated Parallel Recursive Programs

Gopinath Chennupati, R. Muhammad Atif Azad, and Conor Ryan

BDS BIOCOMPUTING AND DEVELOPMENTAL SYSTEMS

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

Programming Multi-Cores

- **Multi-cores** first appearance 1995
- **PCs** and even **Smart Phones** now have multi-cores
- **IBM TrueNorth** 4096 cores
- **SpiNNaker** has in excess of a million processors
  - Biologically Inspired Massively Parallel Architectures
- "If we simply added more than 16 cores, we would get diminishing returns, because the threads and data traffic would not be used properly, so the cores get in the way of each other. It's like having too many cooks in the kitchen."
  - *Jerry Bautista*, director of Intel's tera-scale research program.

Why is parallel programming hard?

- Thread scheduling, synchronization, locking and optimizing the parallelism, etc.

- Efficient parallel programming requires (highly skilled!) human expertise

- ***Automatic Native Parallel Code Generation!***

Human competitive tasks

- Automated the three difficult tasks of humans
  - Optimal parallelism for recursion [1], [3].
  - Automatic architecture awareness [1].
  - Lock-free Programming on multi-cores [2].

[1] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan., (2015) **Performance Optimization of Multi-Core Grammatical Evolution Generated Parallel Recursive Programs**. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO),* edited by Anna I Esparcia Alcázar et al., ACM. In Press.

[2] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan., (2015) **A Multi-Core Grammatical Evolution Based Automatic Lock-Free Programming in OpenMP**. In *Proceedings of the International Conference on Parallel Computing (ParCO),* edited by Gerhard R. Joubert et al., IOS Press. In Press.

[3] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan, (2015) **Automatic Evolution of Parallel Recursive Programs** in Proceedings of EuroGP'15, pages 167 -- 178, Springer.

Criteria

- **D:** The result is publishable in its own right as a new scientific result independent of the fact that the result was mechanically created.
  - [1], [2], [3]

- **E:** The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

- **G:** The result solves a problem of indisputable difficulty in its field.

Recursive Problems

#	Problem	Type		Local Variables	Range
		Input	Output		
1	Sum-of-N	int	int	3	[1, 1000]
2	Factorial	int	unsigned long long	3	[1, 60]
3	Fibonacci	int	unsigned long long	3	[1, 60]
4	Binary-Sum	int [], int, int	int	2	[1, 1000]
5	Reverse	int [], int, int	void	2	[1, 1000]
6	Quicksort	int [], int, int	void	3	[1, 1000]

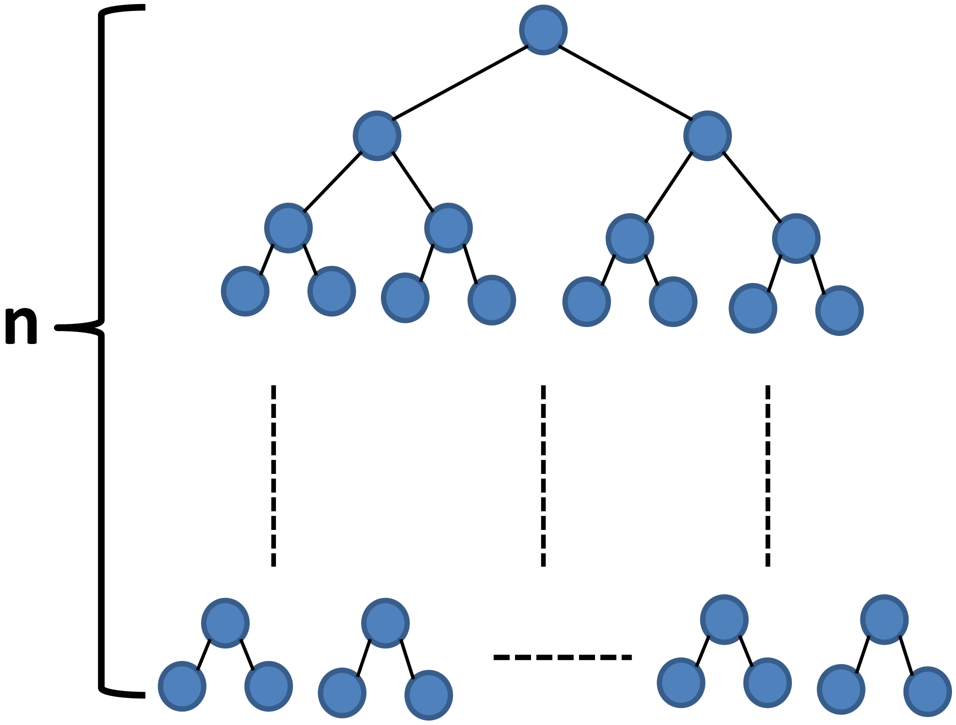
Why Recursion? – *Easy to express but takes longer to execute.*

Excessive Parallelism

Human Program [7]

```
int i, j;

if (n <= 2)      {
          return n;
}
else
{
          #pragma omp parallel sections  \
          shared(i, j)
          {
                    #pragma omp section
                    {
                              i = fib(n−1);
                    }
                    #pragma omp section
                    {
                              j = fib(n−2);
                    }
          }
          return (i+j);
}
```



Maximizing Parallelism

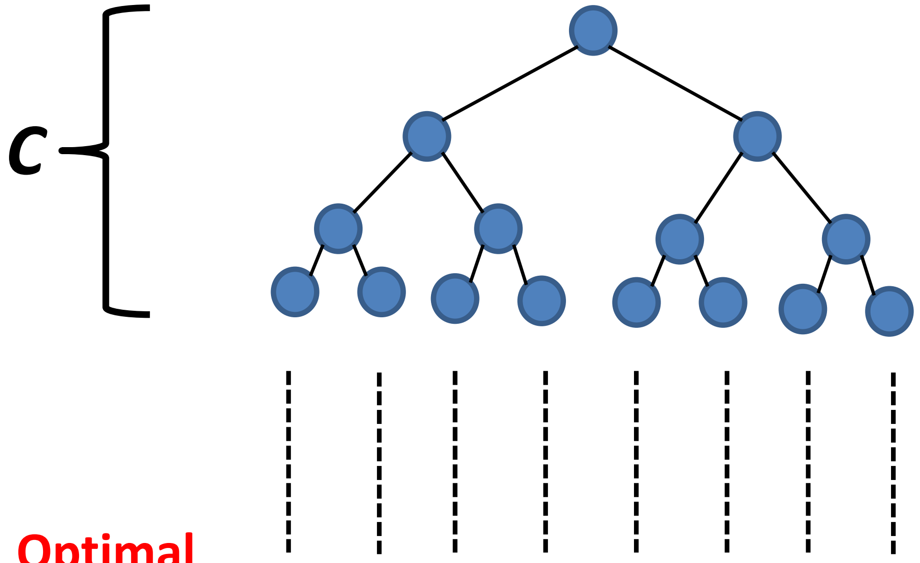
$2^{(n+1)}$ threads

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. (2009) **Introduction to Algorithms, 3rd Edition**. MIT Press.

Optimizing Parallelism

MCGE-II Program

```
if (n <= 2)     {
        temp = n;
        res += temp;
}
else if (n <= 39)     {
        temp = fib(n-1)+fib(n-2);
        res += temp;
}
else     {
        #pragma omp parallel sections  \
        private (a) shared(n, temp, res)
        {
                #pragma omp section
                {
                        a = fib(n−1);
                        #pragma omp atomic
                              res += temp+a;
                }
                #pragma omp section
                {
                        a = fib(n−2);
                        #pragma omp atomic
                              res += temp+a;
                }
        }
} return res;
```

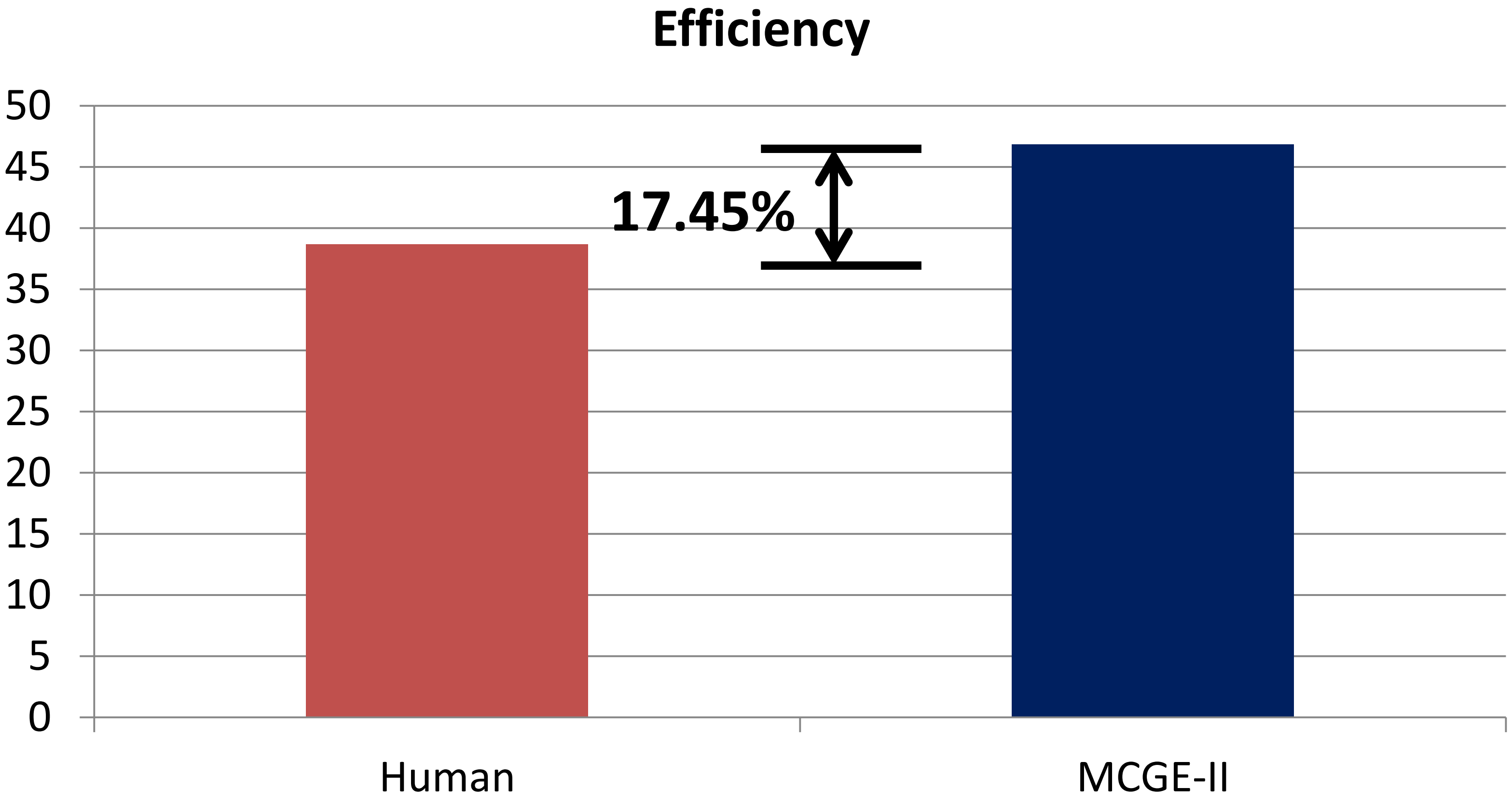


Optimal Parallelism

$2^{(c+1)}$ threads

Satisfies D, G

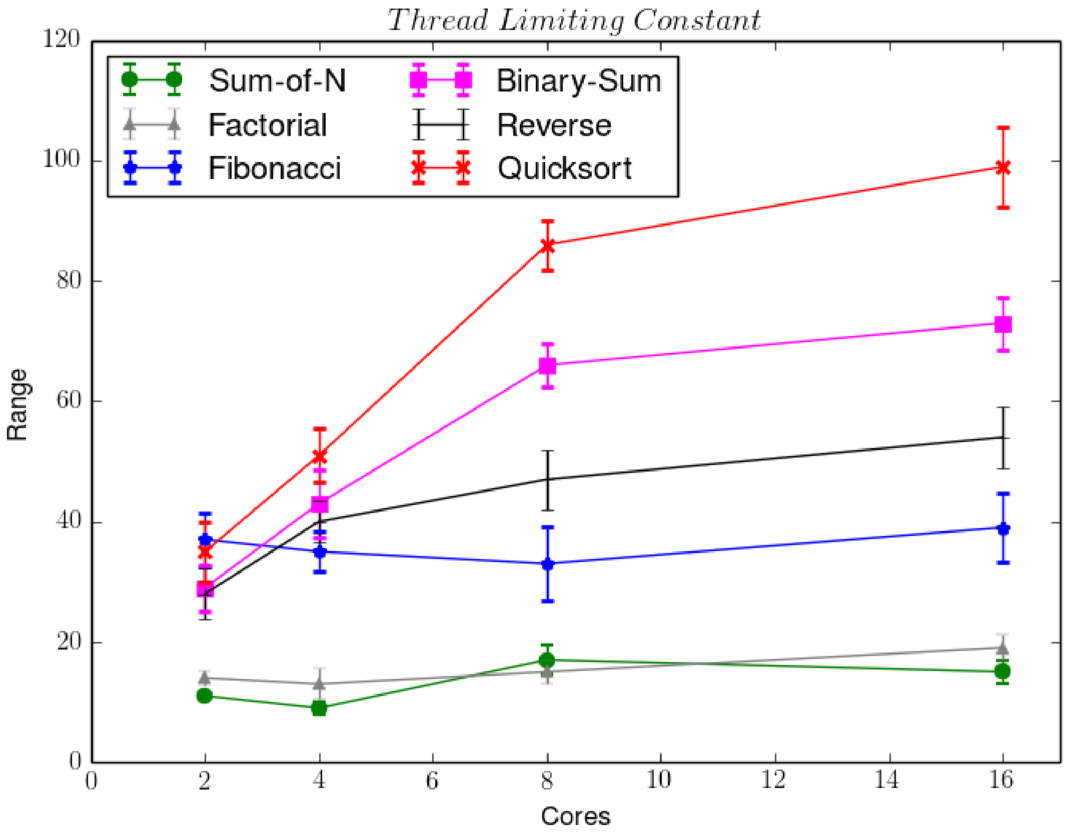
Human Competitive



Satisfies E

Automatic Architecture Awareness

```
if (n <= 2)     {
        temp = n;
        res += temp;
}
else if (n <= 39)     {
        temp = fib(n-1)+fib(n-2);
        res += temp;
}
else      {
        #pragma omp parallel sections  \
        private (a) shared(n, temp, res)
        {
                #pragma omp section
                {
                        a = fib(n−1);
                        #pragma omp atomic
                              res += temp+a;
                }
                #pragma omp section
                {
                        a = fib(n−2);
                        #pragma omp atomic
                              res += temp+a;
                }
        }
} return res;
```



Get it done in 8.35 hours rather waiting forever for humans to figure out!

Satisfies D, G

Lock-Free Parallel Programs

#pragma omp parallel

Lock the shared resources

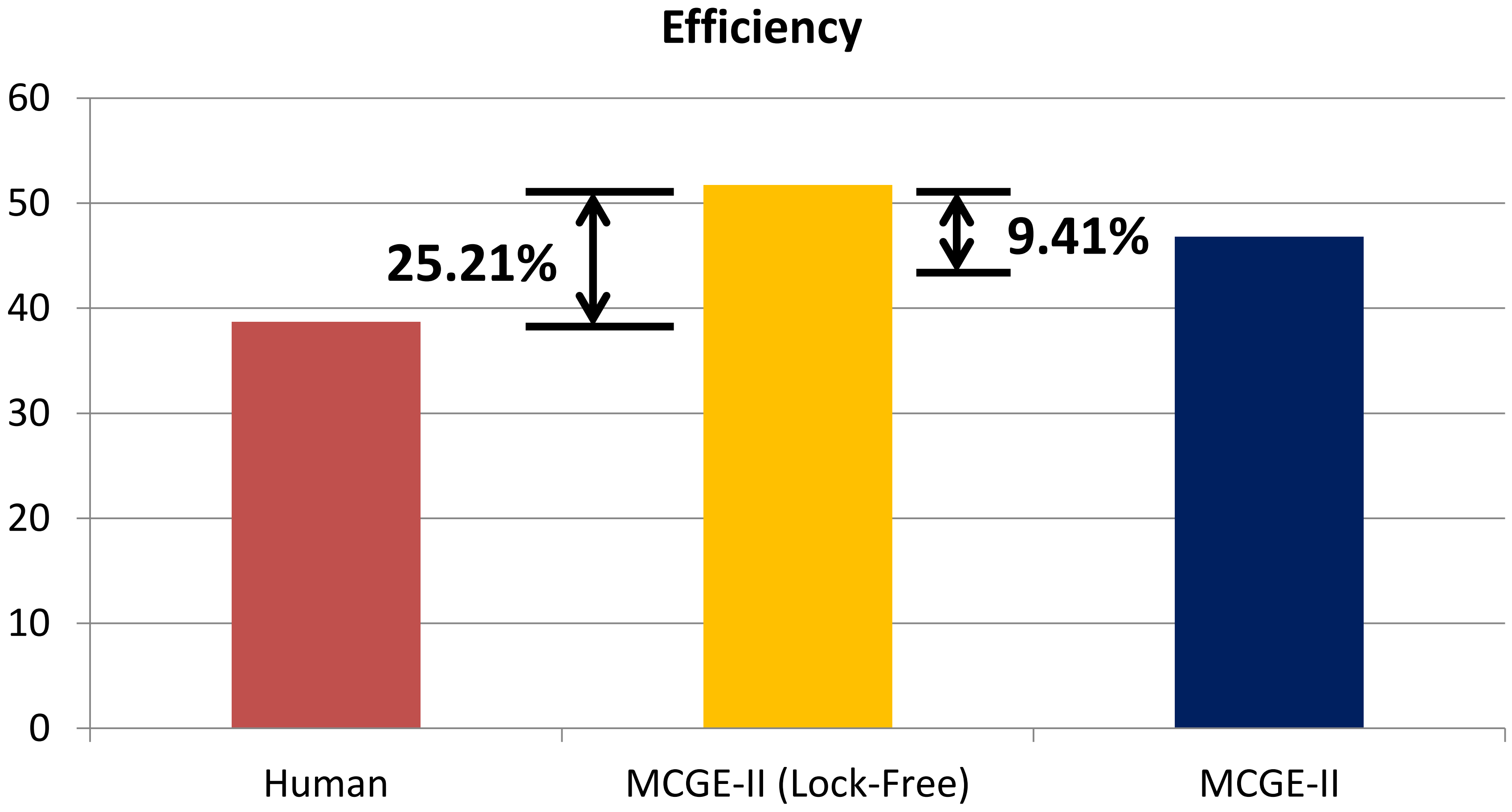
- Locks guarantee mutual exclusion.
- **But**, they degrade the performance.
- Even programming gurus often write ***wrong lock-free programs*** [6].

- **Automatic lock-free parallel programming** [2]

[6] Shane V. Howley and Jeremy Jones. (2012) **A non-blocking internal binary search tree**. In *Proceedings of the 24th annual ACM symposium on Parallelism in algorithms and architectures* (SPAA '12), pages 161--171. ACM

Satisfies D, G

Lock-Free Results



Satisfies E

Potential Impact

- Software
  - Faster to execute parallel code
  - Faster to generate parallel code
- Hardware
  - Better able to utilise multi-core processors
  - Hardware progress (increase in number of cores) less hindered by software limitations

Why we are the best?

- MCGE-II fulfils the original intention of GP as **general purpose** programming tool
- There is an urgent and pressing need in the parallel community for precisely this tool
- The work has been published in a field outside of GP
- This is the first attempt for the synthesis of native parallel programs.

References

[1] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan., (2015) **Performance Optimization of Multi-Core Grammatical Evolution Generated Parallel Recursive Programs**. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO),* edited by Anna I Esparcia Alcázar et al., ACM. In Press.

[2] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan., (2015) **A Multi-Core Grammatical Evolution Based Automatic Lock-Free Programming in OpenMP**. In *Proceedings of the International Conference on Parallel Computing (ParCO),* edited by Gerhard R. Joubert et al., IOS Press. In Press.

[3] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan, (2015) **Automatic Evolution of Parallel Recursive Programs** in Proceedings of EuroGP'15, pages 167 -- 178, Springer.

[4] Gopinath Chennupati, Jeannie Fitzgerald, Conor Ryan, (2014) **On The Efficiency of Multi-core Grammatical Evolution (MCGE) Evolving Multi-Core Parallel Programs** in Proceedings of Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC), pages 238 -- 243, IEEE.

[5] Gopinath Chennupati, R. Muhammad Atif Azad, Conor Ryan, (2014) **Multi-core GE: Automatic Evolution of CPU Based Multi-core Parallel Programs** in Proceedings of GECCO Comp '14, pages 1041 -- 1044, ACM.

[6] Shane V. Howley and Jeremy Jones. (2012) **A non-blocking internal binary search tree**. In *Proceedings of the 24th annual ACM symposium on Parallelism in algorithms and architectures* (SPAA '12), pages 161--171. ACM

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. (2009) **Introduction to Algorithms, 3rd Edition**. MIT Press.