Library for advanced functions in algorithms, data structures and AI implemented in C/C++ – Olib

Biblioteka zaawansowanych funkcji z dziedziny algorytmów i struktur danych oraz AI zaimplementowana w C/C++ - Olib

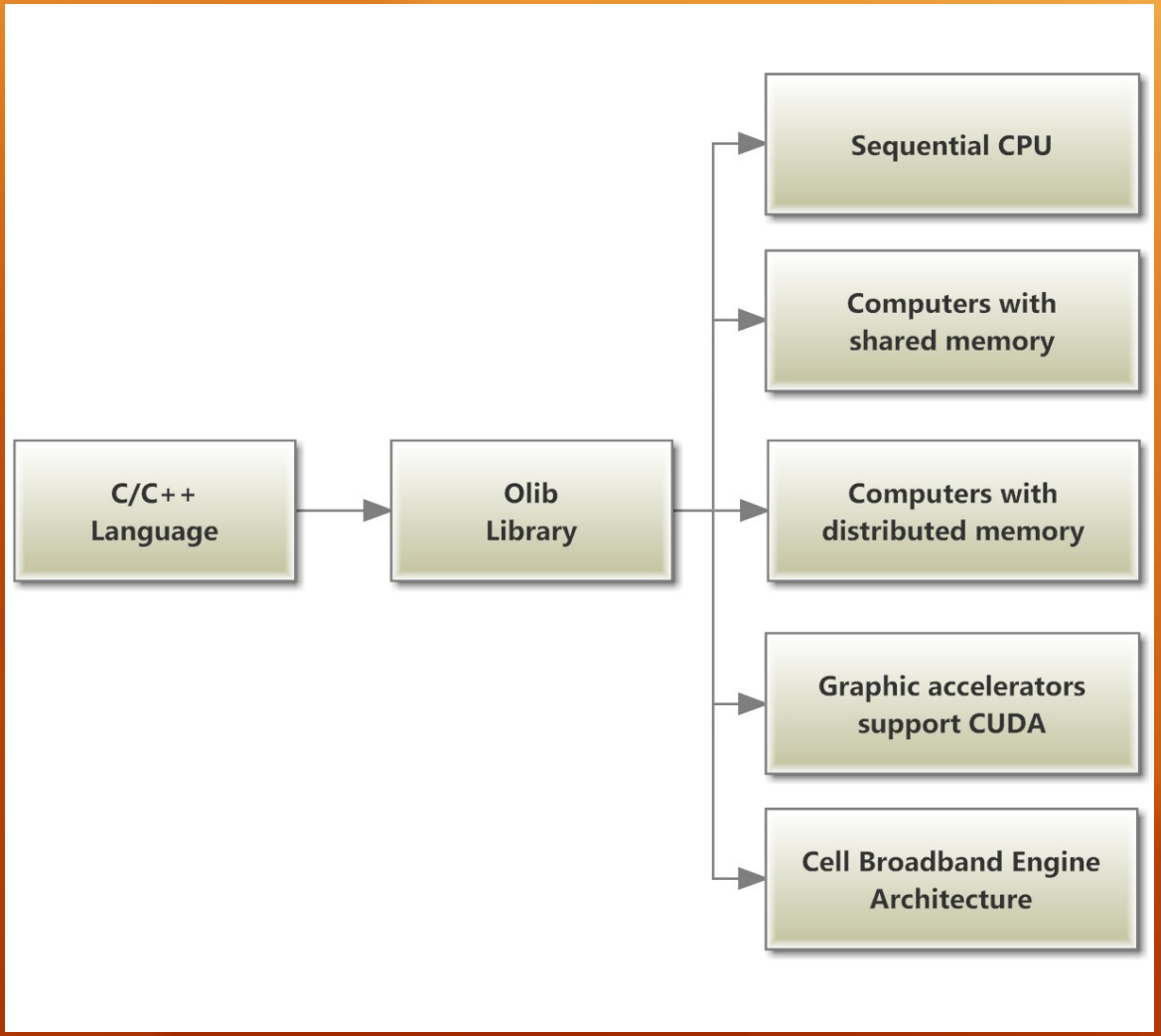
Aleksandra Wszeborowska and Łukasz Świerczewski

Computer Science and Automation Institute, College of Computer Science and Business Administration in Łomża

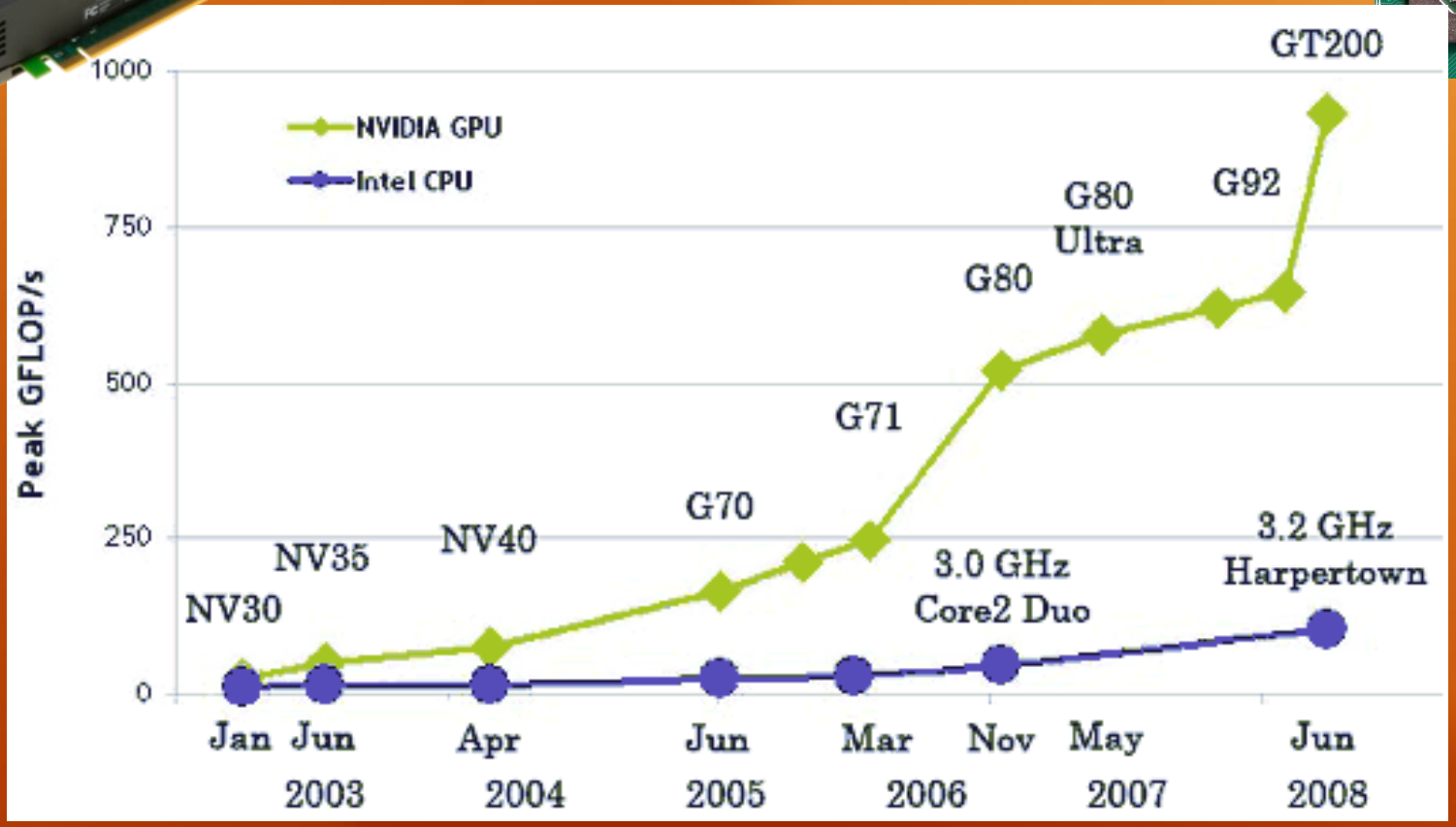
OLib jest biblioteką pisaną głównie z myślą o systemie operacyjnym Linux. W innych systemach (np. Windows, BSD, Solaris) mogą występować drobne problemy ze zgodnością kodu. Zaimplementowano w niej wydajne metody, które można podzielić według działu na:

Algebrę liniową,
Matematykę dyskretną,
Kryptografię,
Metody numeryczne,
Sztuczną inteligencję

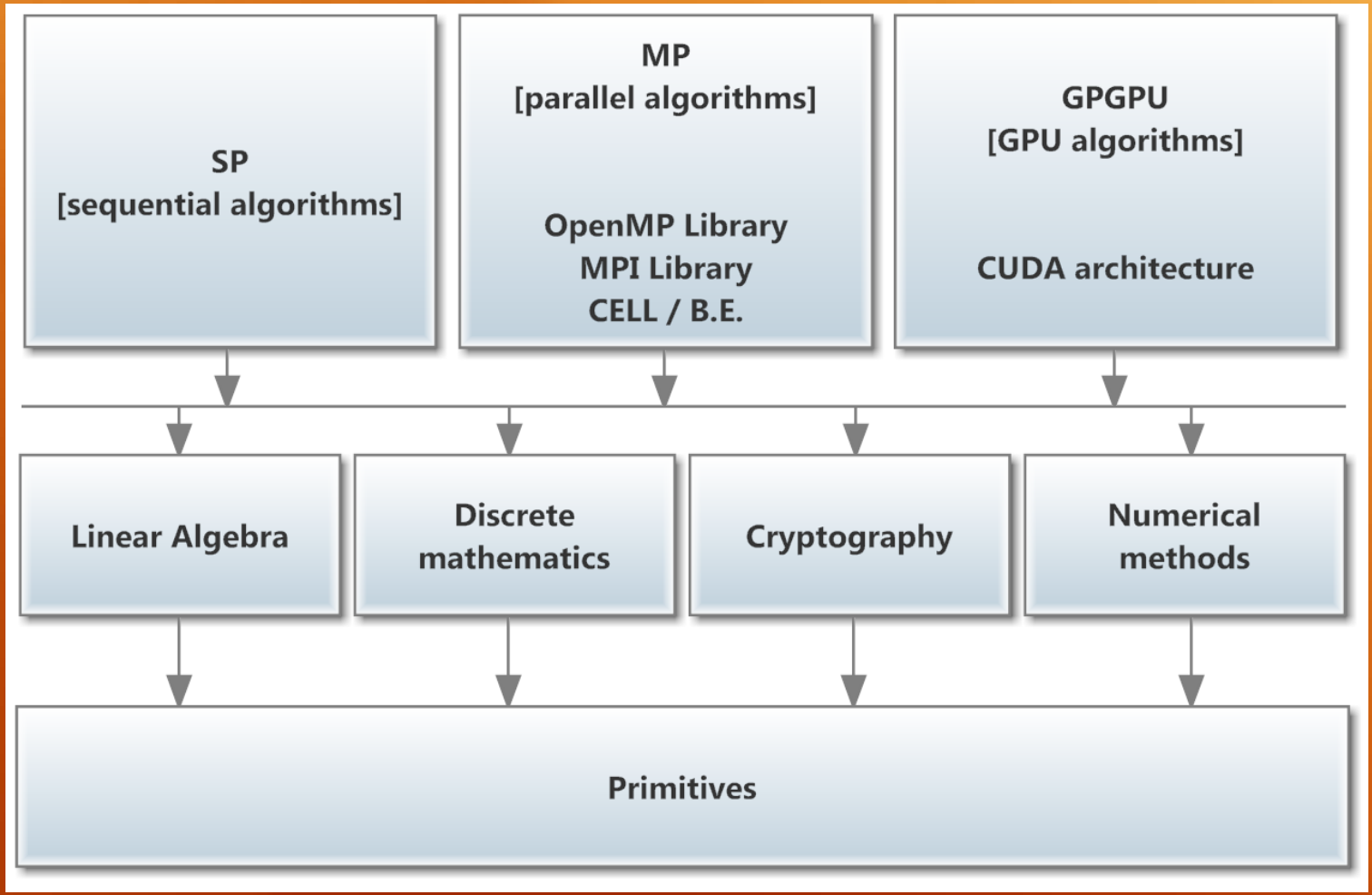
Od strony kryptografii zaimplementowano zarówno szyfry historyczne (m. in. szyfr Cezara, płotkowy, Vigenère'a) jak i te współcześnie stosowane (RSA, DES, 3DES, AES, SHA, MD5). Ze sztuczną inteligencją jest związany głównie Genetic Engine, który daje duże możliwości wykorzystania algorytmów genetycznych do rozwiązywania równań nierozwiązywalnych w sposób analityczny. Genetic Engine cechuje się bardzo dużymi możliwościami zdefiniowania operatorów genetycznych krzyżowania i mutacji, a także innych charakterystyk takich jak metoda selekcji lub rodzaj populacji początkowej.



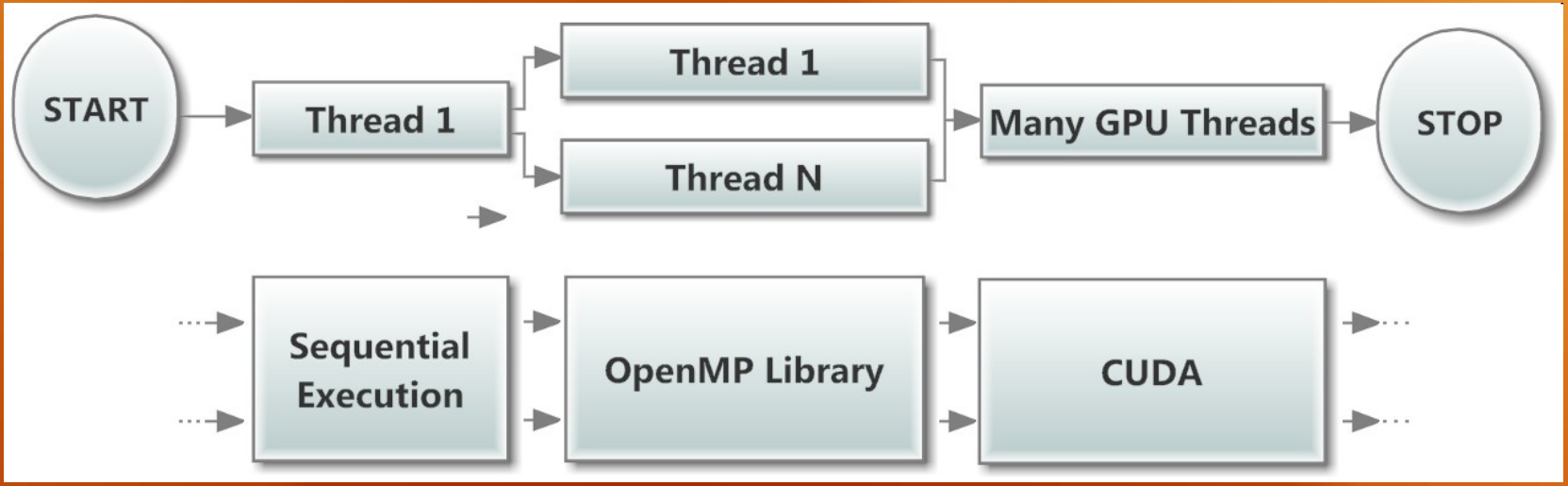
Rysunek 1. *Architektury wspierane przez Olib*



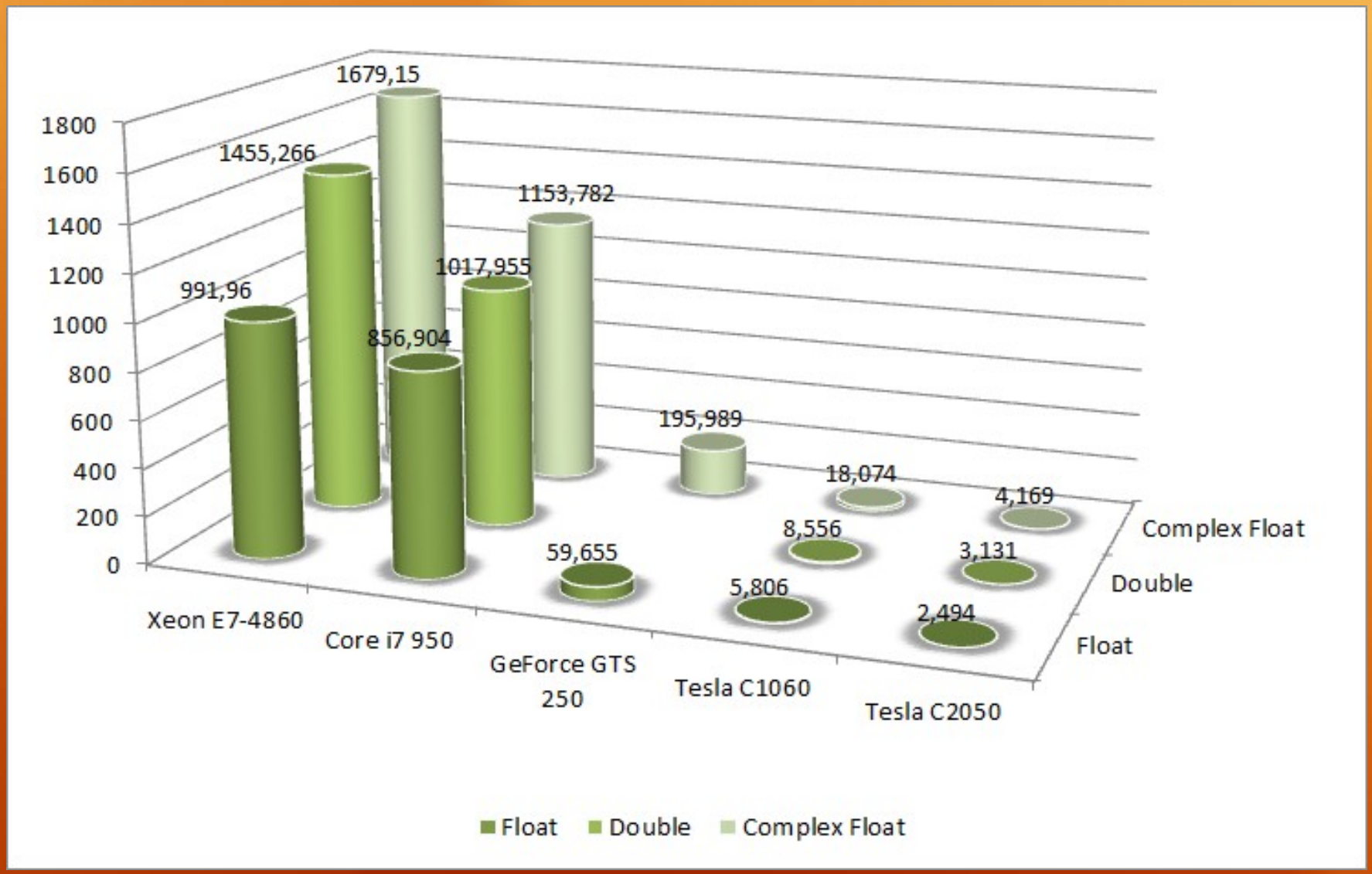
Rysunek 2. Dynamiczny rozwój możliwości obliczeniowych kolejnych generacji GPU w porównaniu do CPU.
Źródło: *Materiały techniczne firmy nVidia*



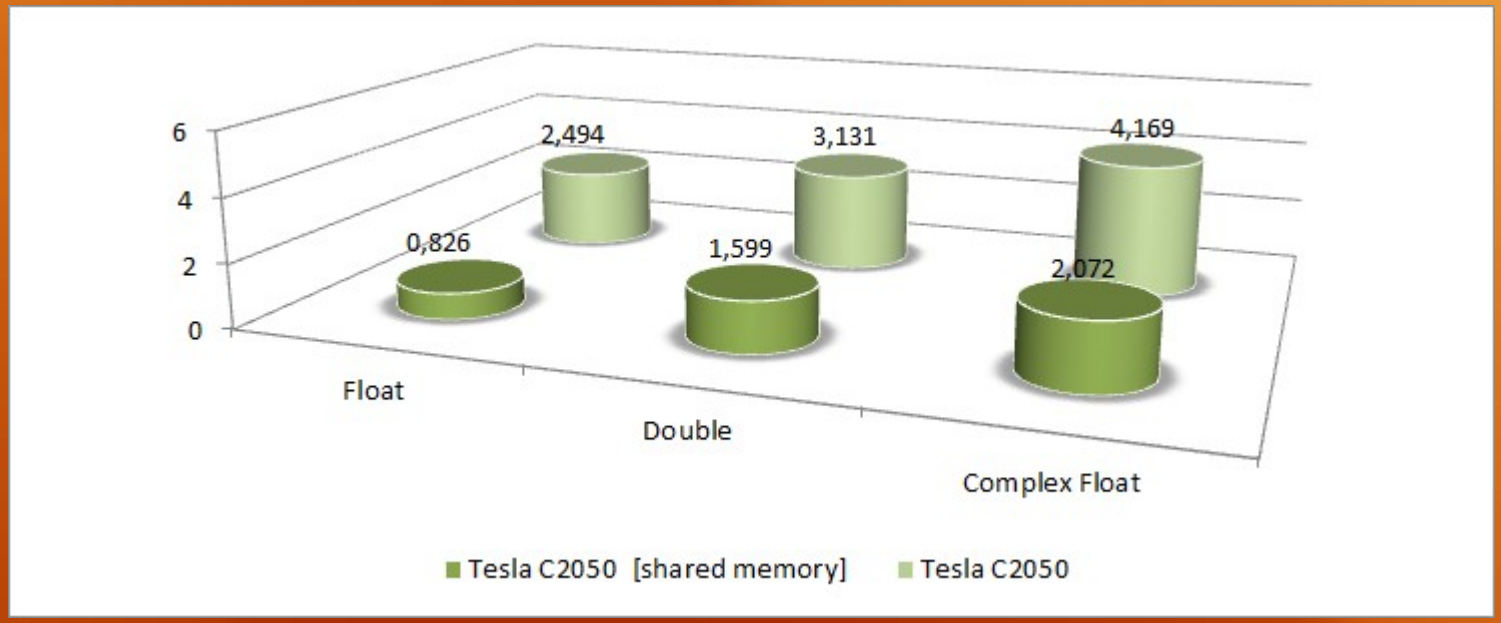
Rysunek 3. *Dostępne moduły*



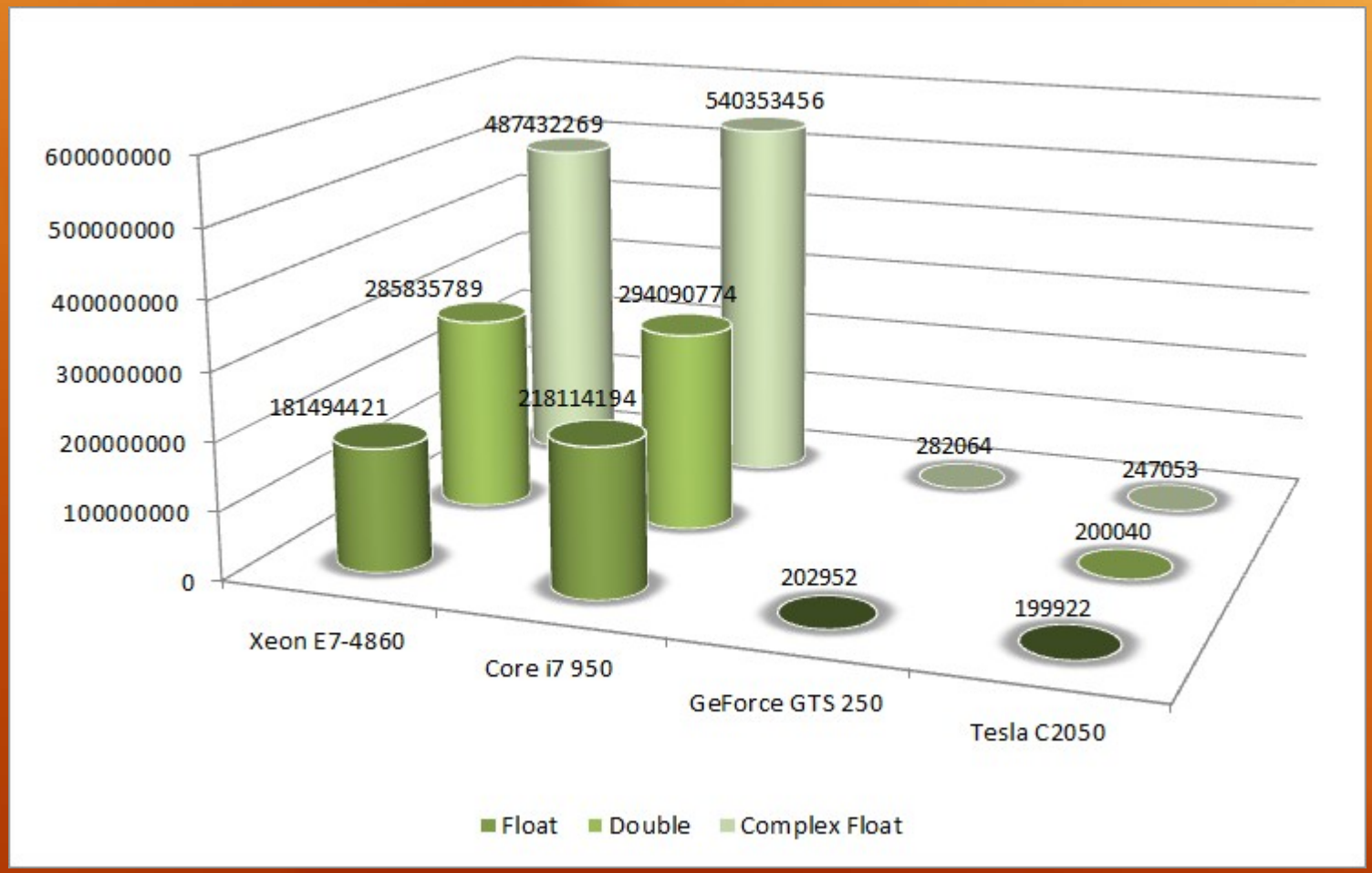
Rysunek 4. *Przykładowy przepływ danych*



Rysunek 5. Czas realizacji (w sekundach) algorytmu mnożenia macierzy z wykorzystaniem różnych układów.
Źródło: Opracowanie własne



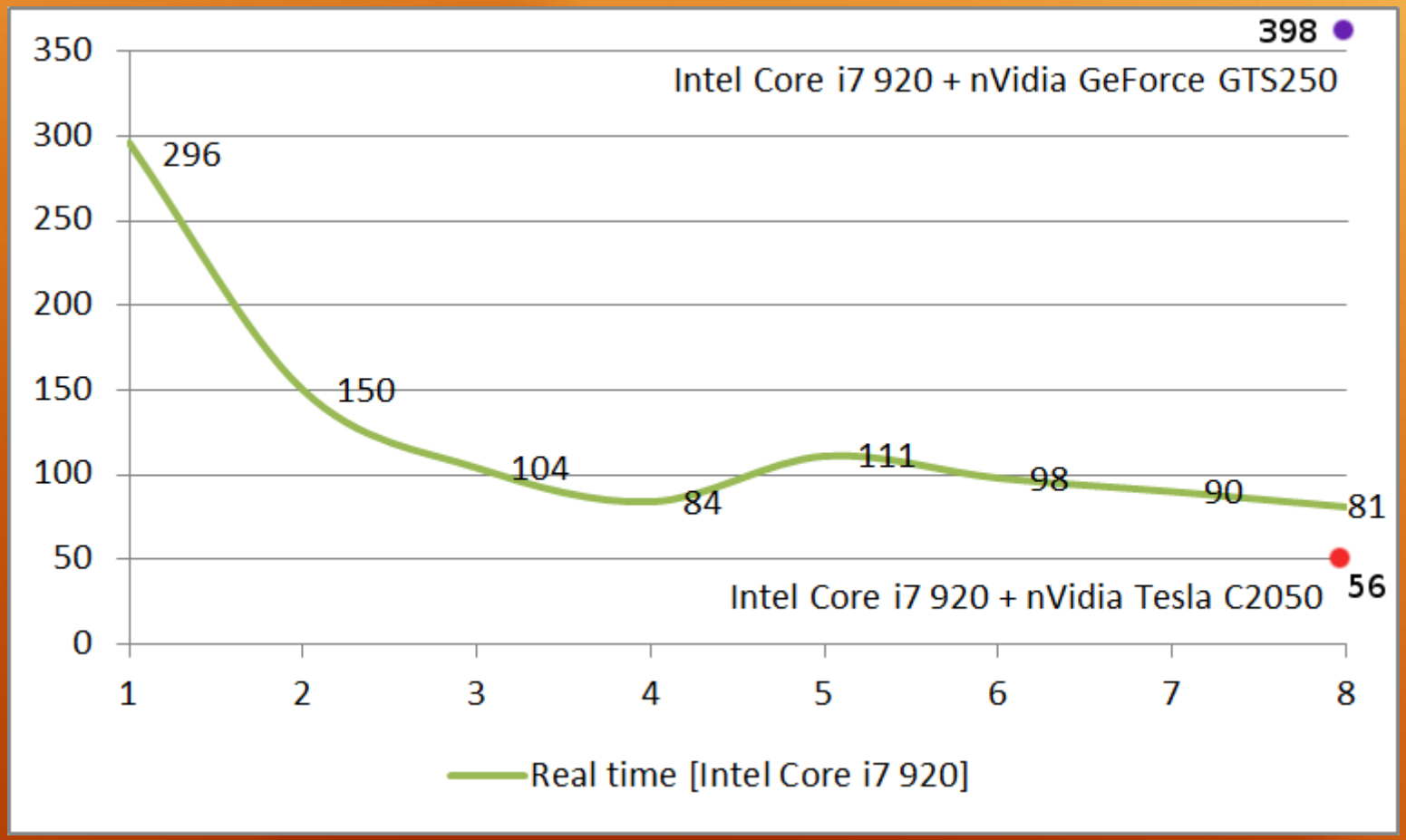
Rysunek 6. Porównanie czasów realizacji algorytmu mnożenia macierzy ze wsparciem i bez wsparcia dla shared memory.
Źródło: Opracowanie własne



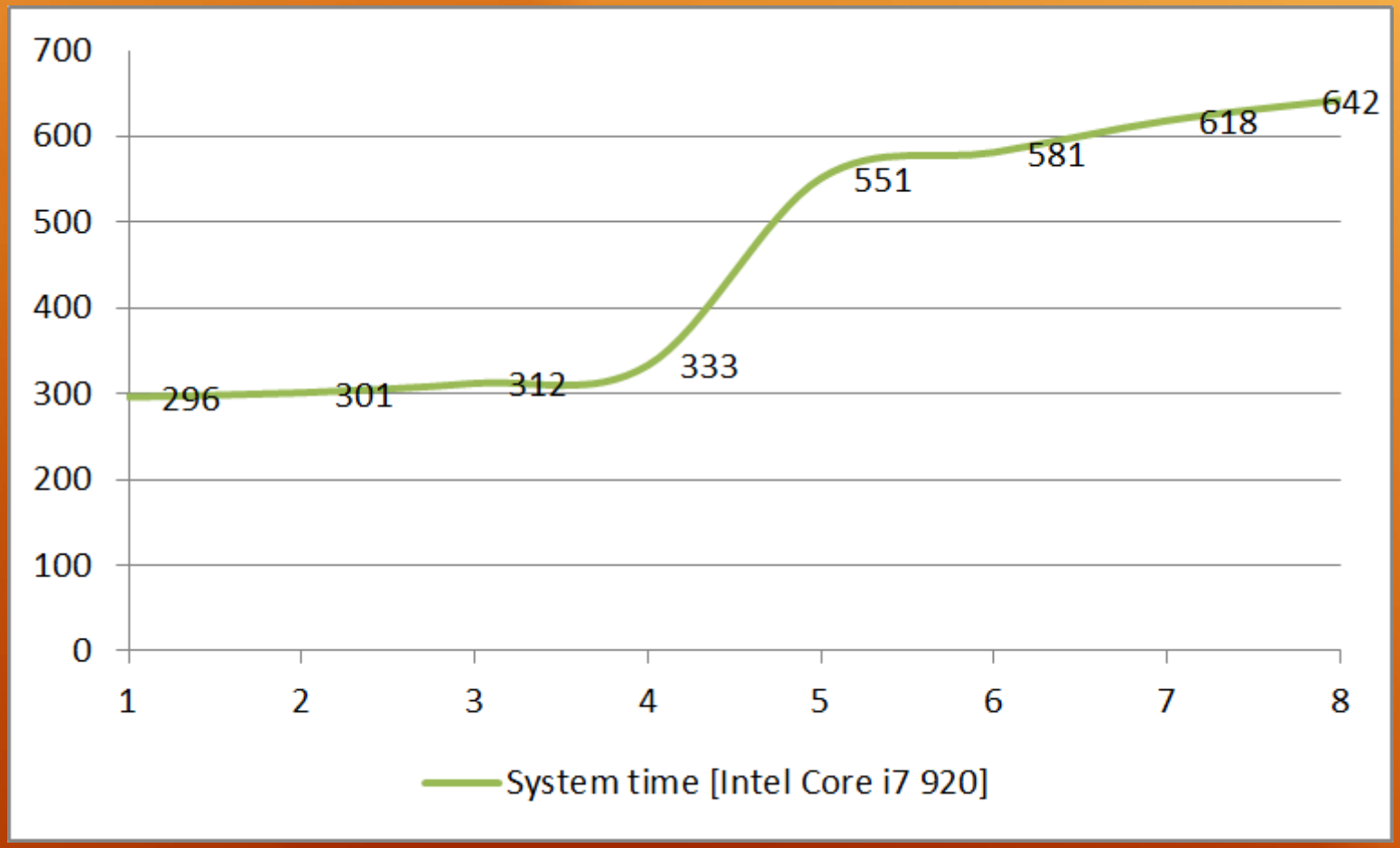
Rysunek 7. Czas realizacji (ilość operacji procesora CPU) algorytmu dodawania / odejmowania macierzy z wykorzystaniem różnych układów.
Źródło: Opracowanie własne

	Float	Przyśpieszenie*	Double	Przyśpieszenie*	Complex Float	Przyśpieszenie*
Intel Xeon E7-4860	991,96	1	1455,27	1	1679,15	1
Intel Core i7 950	856,9	1,16	1017,96	1,43	1153,78	1,46
nVidia GeForce GTS 250	59,66	16,63	-	-	195,99	8,57
nVidia Tesla C1060	5,81	170,85	8,56	170,09	18,07	92,9
nVidia Tesla C2050	2,49	397,74	3,13	464,79	4,17	402,77
nVidia Tesla C2050 [shared memory]	0,83	1195,13	1,6	909,54	2,07	811,18

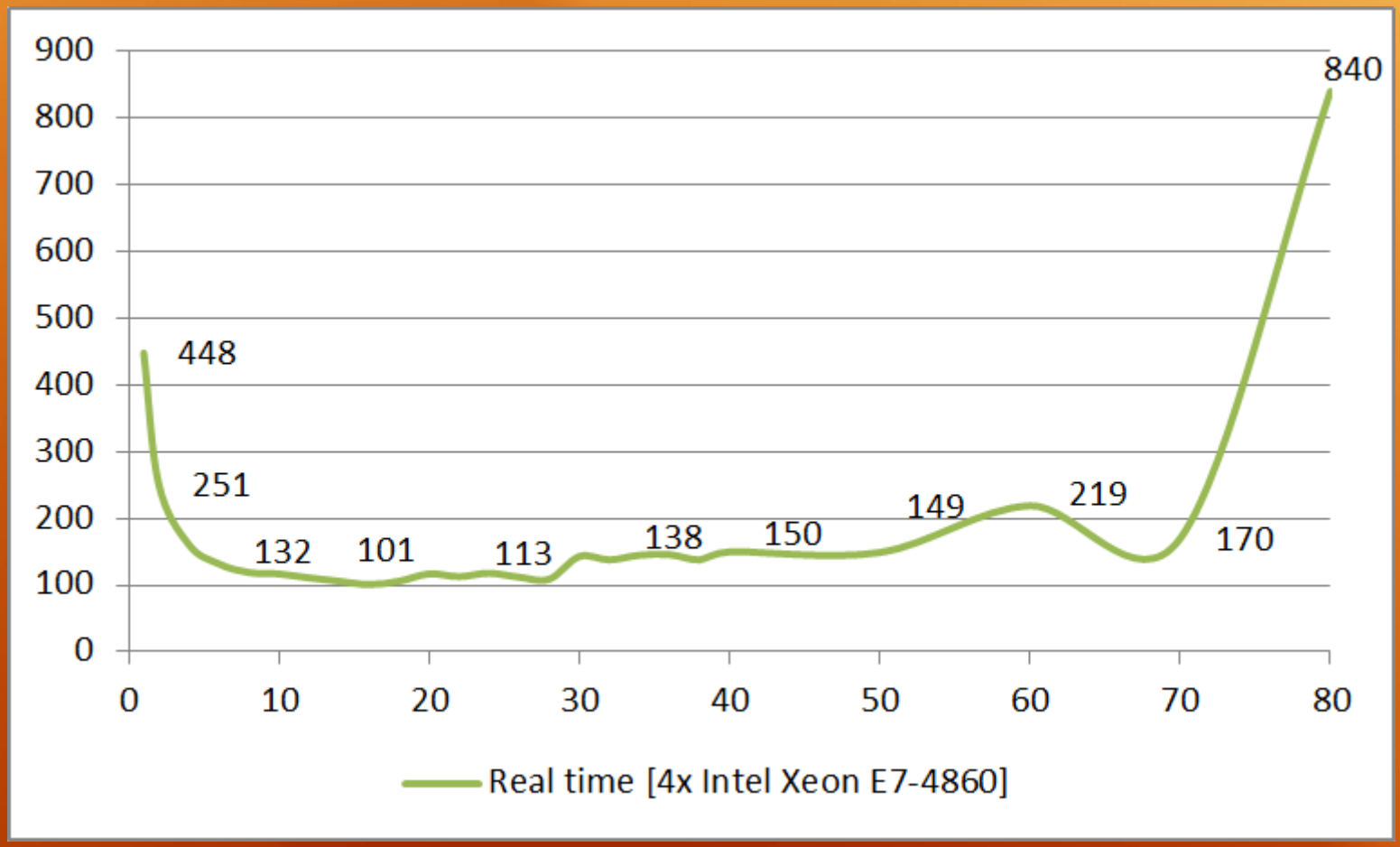
Tabela 1. Prezentacja czasów wykonywania algorytmu mnożenia macierzy (w sekundach) oraz uzyskanych przyśpieszeń.
Źródło: Opracowanie własne



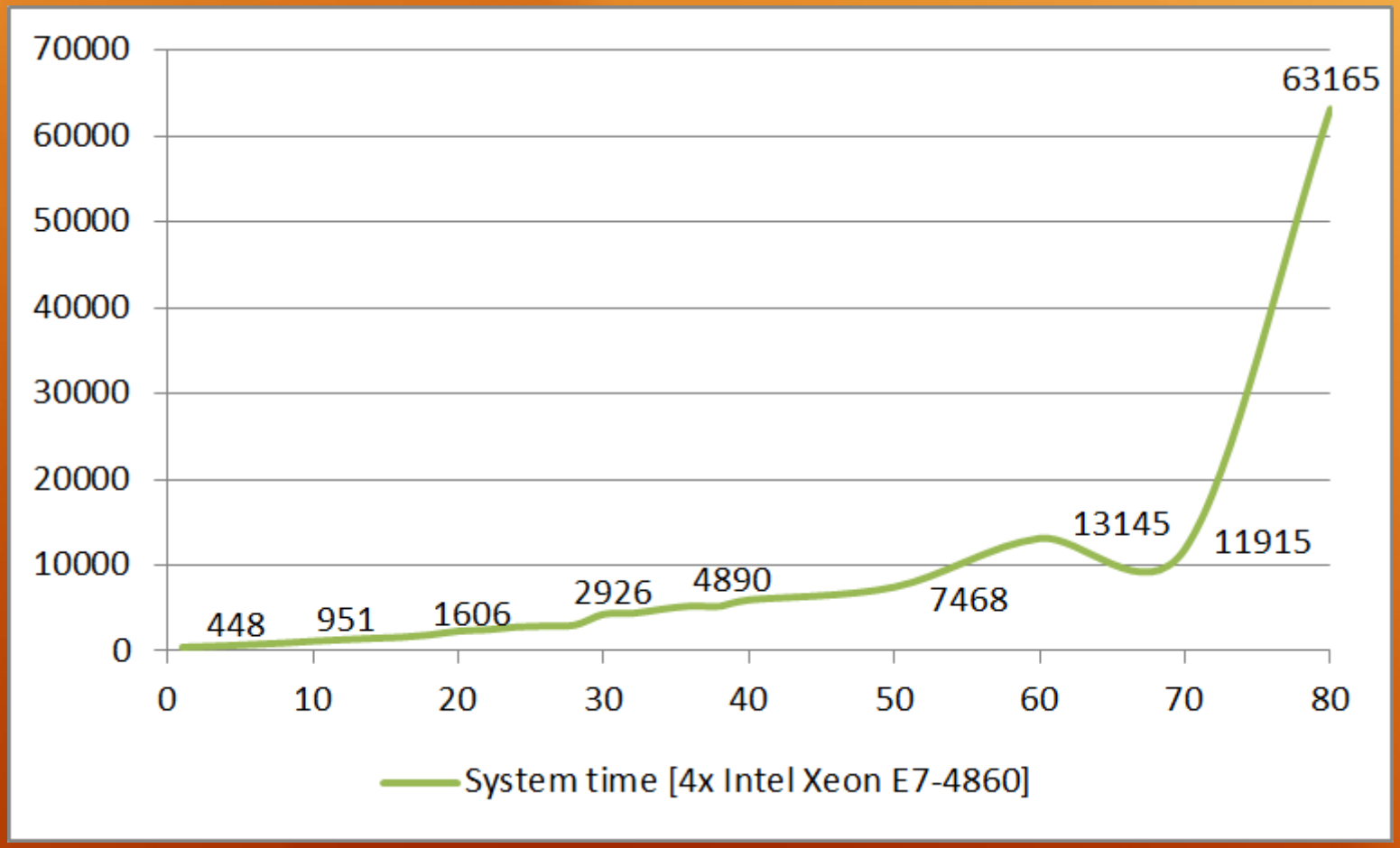
Rysunek 8. *16384 osobników; Bit inversion (P = 0.01); Two-point crossover (P = 0.5); 1000 generations*



Rysunek 9. *16384 osobników; Bit inversion (P = 0.01); Two-point crossover (P = 0.5); 1000 generations*



Rysunek 10. *16384 osobników; Bit inversion (P = 0.01); Two-point crossover (P = 0.5); 1000 generations*



Rysunek 10. *16384 osobników; Bit inversion (P = 0.01); Two-point crossover (P = 0.5); 1000 generations*

CVS Repository:

www.goldbach.pl/olib/

Dziękuję za uwagę ;)