TEST CONDITIONS

Linux 64 bit Core2 Duo 6600 2.4 GHz
Kernel 2.6.37 + GCC 4.5.1 + O3 ffast-math flto fwhole-program static
Type double, secs for 1 mio pure evals, best out of 10 runs

COMPARED PARSERS

MP = muParser 2.1.0
FP = fParser 4.3.0
AT = ATMSP 1.0.0
AL = Acid Library for Microsoft .NET see **SPECIAL NOTES***

MP	FP	AT	AL*	Expression (x=1 y=2 z=3)
0,0103	0,0119	0,0046	3,2760	'3+6
0,0103	0,0120	0,0046	3,2760	5-3
0,0103	0,0119	0,0046	3,3650	'3^2
0,0103	0,0119	0,0046	7,4690	'1-2-3-4
0,0103	0,0119	0,0046	8,6930	'abs(-3)
0,0103	0,0119	0,0046	12,7600	'sin(3)
0,0149	0,0160	0,0112	3,2760	'x+6
0,0220	0,0160	0,0113	3,2760	'x-3
0,0163	0,0126	0,0075	3,3650	'x^2
0,0330	0,0160	0,0250	7,4690	'x-2-3-4
0,0330	0,0238	0,0250	7,4690	'x-y-z-4
0,0290	0,0122	0,0109	8,6930	'abs(-x)
0,0436	0,0436	0,0406	12,7600	'sin(x)
0,0163	0,0168	0,0076		'x^3
0,0163	0,0147	0,0077		'x^4
0,0218	0,0174	0,0142		'x^2+1
0,1100	0,0330	0,0272		'x^4+y^3+z^2
0,0242	0,0125	0,0078		'sqrt(x)
0,0460	0,0151	0,0109		'-sqrt(x)
0,0485	0,0485	0,0485		'sin(2*x)
0,1311	0,1334	0,1308		'sin($pi*x)
0,1388	0,1398	0,1358		'sin($pi*x^2)
0,0935	0,0873	0,0916		'sin($pi^2*x^2)
0,0644	0,0125	0,0142		'abs(-sqrt(x))
0,0878	0,0100	0,0176		'abs(-sqrt(x))^2
0,1143	0,0397	0,0430		'sqrt(abs(-sqrt(x))+2)
0,0879	0,0100	0,0212		'sqrt(abs(-sqrt(x))^4)
0,0297	0,0312	0,0651		'2*(x-y-z-1)
0,0295	0,0295	0,0648		'$pi*(x-y-z-1)
0,0515	0,0340	0,0569		'x+2*y+3*z+1
0,0674	0,0339	0,0576		'x-2*y-3*z-1
0,0473	0,0185	0,0176		'x^2+y^2
0,0229	0,0157	0,0117		'max(2,x)
0,0271	0,0173	0,0151		'max(2,x)^2
0,0907	0,0294	0,0326		'(max(2,x)^2-min(2,y))^2

SPECIAL NOTES for the Acid Library under Microsoft .NET:

Comparing the Acid Library is maybe a bit unfair for various reasons:

1) It is .NET based and Microsoft tells us - more or less subtle – how extra fast this is even in comparison to C++. Especially with floats, what counts here.

2) The Acid benchmarks published where made on a slightly faster PC. Details belov

3) The Acid Library is – quote - "...one of the fastest math parsers available… ...takes advantage of a genuine design… increasing its performance 100 times compared to regular math parsers"

A bit unprivileged by 1) and 2) and a bit anxious by 3) I was curious enough to take the challenge. The conditions published from the vendor of the Acid Library:

Win XP, Athlon 64 X2 Duo 5200+ 2.7 GHz and Microsoft .Net 3.5

Experts may recognize that the Acid Library obviously lacks constant folding. Unfortunately they do not offer other benchmarks. So in favor for Acid the same expressions with variables were added to hinder the other parsers from folding. Do not count beans, but make your own conclusions.