EPSum -- The Problem

- Different results when varying no. of processors
  - Order of operations change with number of processors
  - Precision errors same order of magnitude as programming errors

EPSum – Kahan Sum

- Add carry digits through a second double precision number

	SUM	CORRECTION
	1.0	0.0
+	1.0e-16	
	1.0	1.0e-16
+	1.0e-16	
	1.00000000000000022204	-2.2044604925031313-17

EPSum – The Code

- In practice, the Kahan sum produced as good a result as the Knuth sum at lower cost
  - The setup

```
struct esum_type{
    double sum;
    double correction;
};
int commutative = 1;
MPI_Datatype MPI_TWO_DOUBLES;
MPI_Op KAHAN_SUM;
MPI_Type_contiguous(2, MPI_DOUBLE, &MPI_TWO_DOUBLES);
MPI_Type_commit(&MPI_TWO_DOUBLES);
MPI_Op_create((MPI_User_function *)kahan_sum, commutative, &KAHAN_SUM);
```

  - And the cleanup

```
MPI_Op_free(&KAHAN_SUM);
MPI_Type_free(&MPI_TWO_DOUBLES);
```

EPSum – The Code

- The MPI Operation

```
void kahan_sum(struct esum_type *in, struct esum_type *inout, int *len,
    MPI_Datatype *MPI_TWO_DOUBLES)
{
    double corrected_next_term, new_sum;
    corrected_next_term = in->sum + (in.correction+inout.correction);
    new_sum = inout->sum + corrected_next_term;
    inout->correction = corrected_next_term - (new_sum - inout->sum);
    inout->sum = new_sum;
}
```

EPSum – The Code

```
double simple_parallel_kahan_sum(double **var)
{
    double corrected_next_term, new_sum;
    struct esum_type local, global;
    local.sum=0.0;
    local.correction=0.0;
    for(j=0; j<jsize; j++){
        for(i=0; i<isize; i++){
            corrected_next_term = var[j][i] + local.correction;
            new_sum = local.sum + corrected_next_term;
            local.correction = corrected_next_term - (new_sum-local.sum);
            local.sum = new_sum;
        }
    }
    MPI_Allreduce(&local, &global, 1, MPI_TWO_DOUBLES, KAHAN_SUM, MPI_COMM_WORLD);
    return(global.sum);
}
```

EPSum -- Results

Proc	Normal	Long-Double	Kahan	Knuth
1	-68009	63.017	0	0
2	-51316	33.374	-2	-2
4	-32740	-3.325	2	2
8	2586	-3.004	-2	-2
16	2360	-2.500	-2	-2
32	1968	-1.621	-2	-2
64	1078	0.134	-4	-4
128	-726	-1.031	-2	-2
256	192	-0.998	-2	-2

*In multiples of machine epsilon