# Implementing a NRG code, handling second quantization expressions, symmetries, parallelization issues

Rok Žitko
Institute Jožef Stefan
Ljubljana, Slovenia

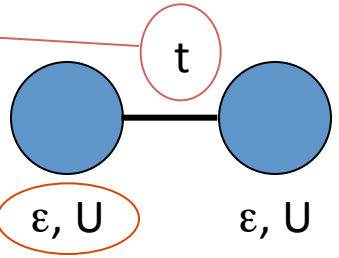
# Tools: SNEG and NRG Ljubljana

Add-on package for the computer algebra system Mathematica for performing calculations involving non-commuting operators

Efficient general purpose numerical renormalization group code

- flexible and adaptable
- highly optimized (partially parallelized)
- easy to use

```
In[24]:= H = Sum[e number[c[i]] + U hubbard[c[i]], {i, 2}] + t hop[c[1], c[2]]
         s = Sum[spinz[c[i]], {i, 2}]
         komutator[H, s] // Expand
         n = Sum[number[c[i]], {i, 2}]
         komutator[H, n] // Expand
```




$$\text{Out[24]}= e\left(c_{1\downarrow}^{\dagger}\cdot c_{1\downarrow}+c_{1\uparrow}^{\dagger}\cdot c_{1\uparrow}\right)+t\left(c_{1\downarrow}^{\dagger}\cdot c_{2\downarrow}+c_{1\uparrow}^{\dagger}\cdot c_{2\uparrow}+c_{2\downarrow}^{\dagger}\cdot c_{1\downarrow}+c_{2\uparrow}^{\dagger}\cdot c_{1\uparrow}\right)+ e\left(c_{2\downarrow}^{\dagger}\cdot c_{2\downarrow}+c_{2\uparrow}^{\dagger}\cdot c_{2\uparrow}\right)-U\,c_{1\downarrow}^{\dagger}\cdot c_{1\uparrow}^{\dagger}\cdot c_{1\downarrow}\cdot c_{1\uparrow}-U\,c_{2\downarrow}^{\dagger}\cdot c_{2\uparrow}^{\dagger}\cdot c_{2\downarrow}\cdot c_{2\uparrow}$$

$$\text{Out[25]}= \frac{1}{2}\left(-c_{1\downarrow}^{\dagger}\cdot c_{1\downarrow}+c_{1\uparrow}^{\dagger}\cdot c_{1\uparrow}\right)+\frac{1}{2}\left(-c_{2\downarrow}^{\dagger}\cdot c_{2\downarrow}+c_{2\uparrow}^{\dagger}\cdot c_{2\uparrow}\right)$$

$$\text{Out[26]}= 0$$

$$\text{Out[27]}= c_{1\downarrow}^{\dagger}\cdot c_{1\downarrow}+c_{1\uparrow}^{\dagger}\cdot c_{1\uparrow}+c_{2\downarrow}^{\dagger}\cdot c_{2\downarrow}+c_{2\uparrow}^{\dagger}\cdot c_{2\uparrow}$$

$$\text{Out[28]}= 0$$

```
In[33]:= bvc = qsbasisvc[{c[1], c[2]}]; bvc // MatrixForm
```

Out[33]//MatrixForm=

| | |
|---|---|
| {-2, 0} | {\|□ □ □ □⟩} |
| {-1, 1/2} | {\|□ □ ■ □⟩, \|■ □ □ □⟩} |
| {0, 0} | {-\|□ □ ■ ■⟩, (\|□ ■ □ ■⟩-\|■ □ ■ □⟩)/√2, -\|■ ■ □ □⟩} |
| {0, 1} | {-\|■ □ ■ □⟩} |
| {1, 1/2} | {-\|■ □ ■ ■⟩, -\|■ ■ ■ □⟩} |
| {2, 0} | {\|■ ■ ■ ■⟩} |

```
In[36]:= m = matrixrepresentationvc[H, bvc[[3,2]]]; m // MatrixForm
```

Out[36]//MatrixForm=

$$\begin{pmatrix} U+2e & \sqrt{2}\,t & 0 \\ \sqrt{2}\,t & 2e & \sqrt{2}\,t \\ 0 & \sqrt{2}\,t & U+2e \end{pmatrix}$$

# SNEG - features

- **fermionic** (Majorana, Dirac) and **bosonic** operators, **Grassman** numbers
- basis construction (well defined number and spin (*Q*,*S*), isospin and spin (*I*,*S*), etc.)
- **symbolic sums** over dummy indexes (***k***, $\sigma$)
- **Wick's theorem** (with either empty band or Fermi sea vacuum states)
- Dirac's **bra** and **ket** notation
- Simplifications using **Baker-Campbell-Hausdorff** and **Mendaš-Milutinović** formula

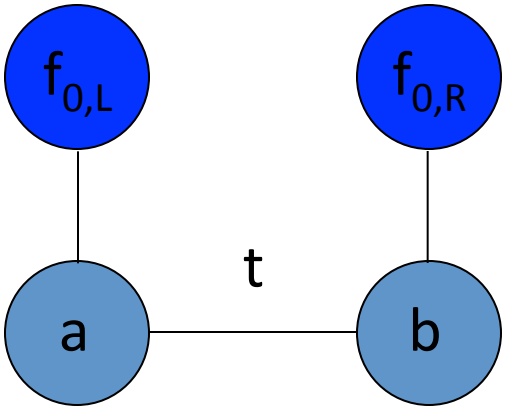
# SNEG - applications

- exact diagonalization of small clusters
- perturbation theory to high order
- high-temperature series expansion
- evaluation of (anti-)commutators of complex expressions
- NRG
  - derivation of coefficients required in the NRG iteration
  - problem setup

# “NRG Ljubljana” - goals

- Flexibility (very few hard-coded limits, adaptability)
- Implementation using modern high-level programming paradigms
  (functional programming in Mathematica,
  object oriented programming in C++)
  $\Rightarrow$ short and maintainable code
- Efficiency (LAPACK routines for diagonalization)
- Free availability

# Definition of a quantum impurity problem in "NRG Ljubljana"

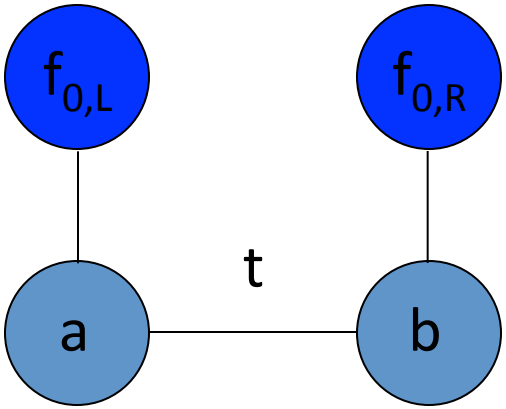



```
Himp = eps (number[a[]]+number[b[]])+
U/2 (pow[number[a[]]-1,2]+pow[number[b[]]-1,2])

Hab = t hop[a[],b[]] + V change[a[]b[]]

Hc = Sqrt[Gamma] (hop[a[],f[L]] + hop[b[],f[R]])
```

# Definition of a quantum impurity problem in "NRG Ljubljana"



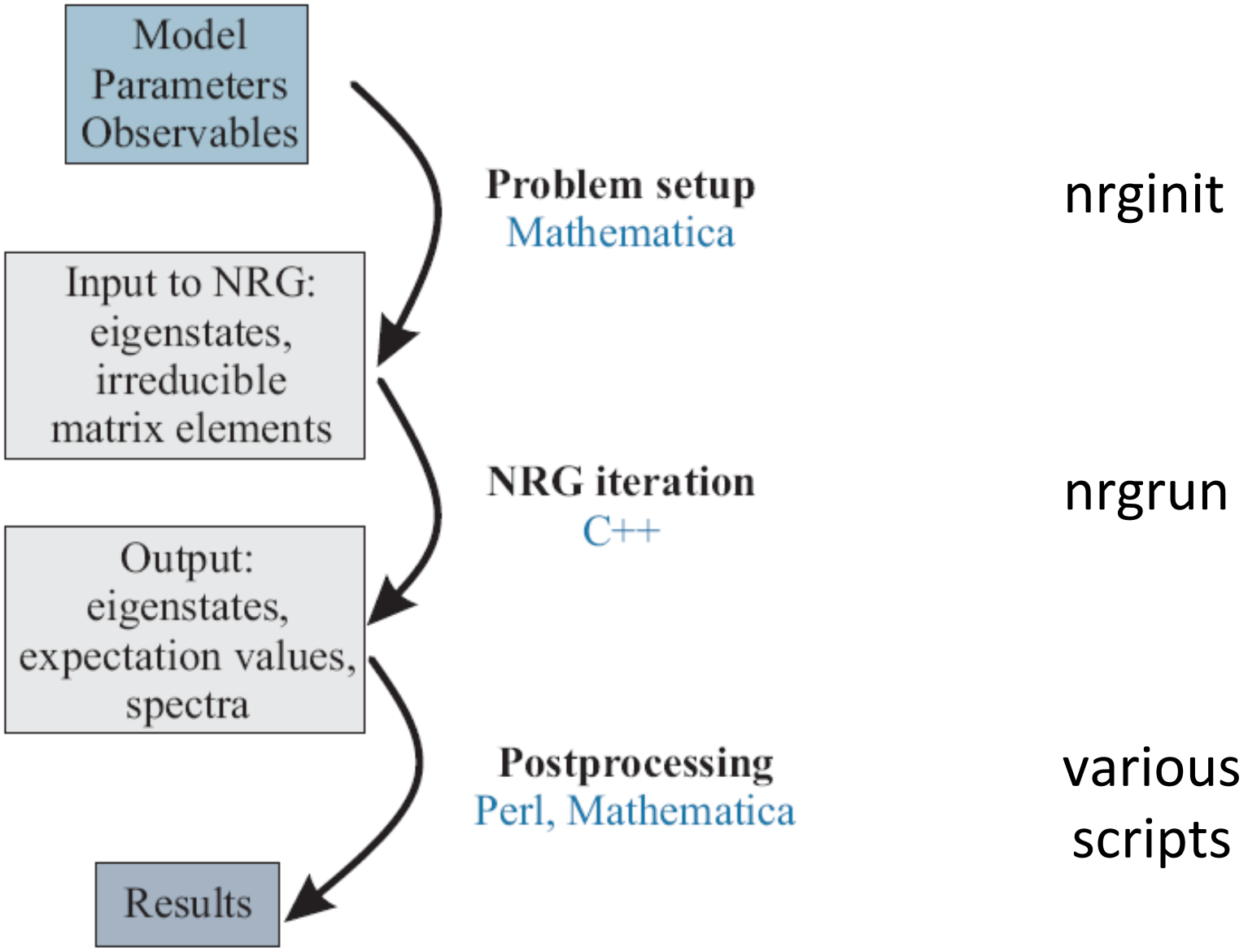

```
Himp = epsa number[a[]] + epsb number[b[]] +
U/2 (pow[number[a[]]-1,2]+pow[number[b[]]-1,2])

Hab = t hop[a[],b[]]

Hc = Sqrt[Gamma] (hop[a[],f[L]] + hop[b[],f[R]])
```

*“By relieving the brain of all unnecessary work, a **good notation** sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.”*

Alfred North Whitehead


Model
Parameters
Observables
Problem setup
Mathematica
nrginit
Input to NRG:
eigenstates,
irreducible
matrix elements
NRG iteration
C++
nrgrun
Output:
eigenstates,
expectation values,
spectra
Postprocessing
Perl, Mathematica
various
scripts
Results

# Computable quantities

- **Finite-site excitation spectra** (flow diagrams)
- **Thermodynamics**:
  magnetic and charge susceptibility, entropy, heat capacity
- **Correlations**:
  spin-spin correlations, charge fluctuations,...
  `spinspin[a[],b[]]`
  `number[d[]]`
  `pow[number[d[]], 2]`
- **Dynamics**:
  spectral functions, dynamical magnetic and charge susceptibility, other response functions

# Sample input file

```
[param]
model=SIAM
U=1.0
Gamma=0.04

Lambda=3
Nmax=40
keepenergy=10.0
keep=2000

ops=q_d q_d^2 A_d
```

Model and parameters

NRG iteration parameters

Computed quantities

Spectral function

Charge fluctuations

Occupancy

# Spin symmetry $\mathrm{SU}(2)_{\mathrm{spin}}$

$$\mathbf{S}_i = \sum_{\mu\mu'} a^\dagger_{i\mu} \left(\tfrac{1}{2}\boldsymbol{\sigma}_{\mu\mu'}\right) a_{i\mu'} \qquad \boldsymbol{\sigma} = \{\sigma^x, \sigma^y, \sigma^z\}$$

$$S^z_i = \tfrac{1}{2}\left(a^\dagger_{i\uparrow}a_{i\uparrow} - a^\dagger_{i\downarrow}a_{i\downarrow}\right)$$

$$S^+_i = a^\dagger_{i\uparrow}a_{i\downarrow} \qquad S^+_i = S^x + iS^y$$

$$S^-_i = a^\dagger_{i\downarrow}a_{i\uparrow} \qquad S^-_i = S^x - iS^y.$$

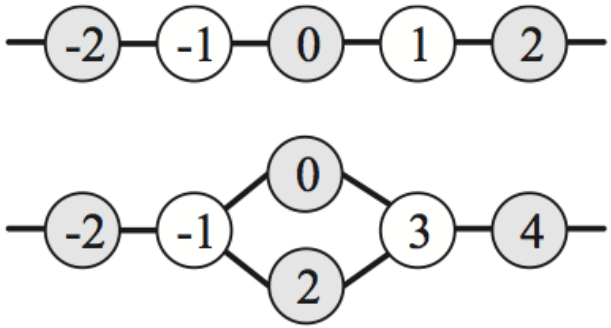
$$\mathbf{S} = \sum_i \mathbf{S}_i$$

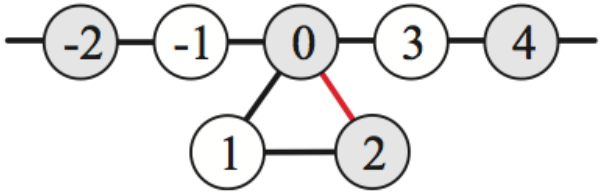
# Charge and particle-hole symmetry

$$\hat{Q} = \sum_i (n_i - 1)$$

$$a^\dagger_{i,-\mu} \rightarrow (-1)^i (2\mu) a_{i,-\mu} \qquad A(\omega) = A(-\omega)$$

$$t\left(a_i^\dagger a_{i+1} + a_{i+1}^\dagger a_i\right) \rightarrow t\left((-1)^i a_i (-1)^{i+1} a_{i+1}^\dagger + (-1)^{i+1} a_{i+1} (-1)^i a_i^\dagger\right) = t\left(a_i^\dagger a_{i+1} + a_{i+1}^\dagger a_i\right)$$



# Isospin symmetry $\mathrm{SU(2)_{iso}}$

Nambu spinor: $\eta_{i,\mu} = \begin{pmatrix} a^\dagger_{i,\mu} \\ (-1)^i (2\mu) a_{i,-\mu} \end{pmatrix}$ $\xi^\dagger_i = \begin{pmatrix} a^\dagger_{i,\uparrow} \\ (-1)^i a_{i,\downarrow} \end{pmatrix}$

Izospin operator: $\mathbf{I}_i = \xi_i \left(\frac{1}{2}\boldsymbol{\sigma}\right) \xi^\dagger_i$

$$I^z_i = \tfrac{1}{2}\left(a^\dagger_{i,\uparrow} a_{i,\uparrow} + a^\dagger_{i,\downarrow} a_{i,\downarrow} - 1\right) = \tfrac{1}{2} Q_i \longleftarrow \text{charge}$$

$$I^+_i = (-1)^i a^\dagger_{i,\uparrow} a^\dagger_{i,\downarrow}$$

$$I^-_i = (-1)^i a_{i,\downarrow} a_{i,\uparrow}. \longleftarrow \text{pairing}$$

$$[I^z_i, a^\dagger_{j,\mu}] = \delta_{ij} \tfrac{1}{2} a^\dagger_{i,\mu}$$

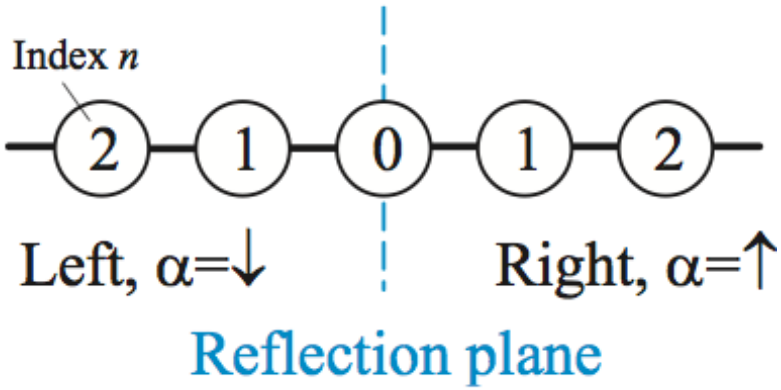
$$[I^z_i, a_{j,\mu}] = \delta_{ij} (-\tfrac{1}{2}) a_{i,\mu}$$

$$[I^-_i, a^\dagger_{j,\mu}] = \delta_{ij} (-1)^i (2\mu) a_{i,-\mu}$$

$$[I^+_i, (-1)^j (2\mu) a_{j,-\mu}] = \delta_{ij} a^\dagger_{i,\mu}.$$

# Reflection symmetry

Parity $Z_2$ quantum number P




"Flavor symmetry" $SU(2)_{flavor}$:

$$\mathbf{J}_n = \sum_{\mu} \sum_{\alpha\alpha'} a^{\dagger}_{n\alpha\mu} \left(\tfrac{1}{2}\boldsymbol{\sigma}_{\alpha\alpha'}\right) a_{n\alpha'\mu}$$

# Wigner-Eckart theorem

O is a **spherical tensor operator** of rank M if:

$$\left[J_z, O_\mu^M\right] = \mu O_\mu^M$$
$$\left[J_+, O_\mu^M\right] = A(M,\mu) O_{\mu+1}^M$$
$$\left[J_-, O_\mu^M\right] = A(M,-\mu) O_{\mu-1}^M$$

$$A(M,\mu) = \sqrt{(M-\mu)(M+\mu+1)}$$

$$\langle \alpha, j, j_z | O_\mu^M | \alpha', j', j_z' \rangle = \langle j' j_z'; M\mu | j j_z \rangle \langle \alpha, j \| O^M \| \alpha', j' \rangle$$

Clebsch-Gordan coefficients for SU(2)

For a more general treatment of non-Abelian symmetries in NRG, see
A. I. Toth, C. P. Moca, O. Legeza, G. Zarand, PRB 78, 245109 (2008),
A. Weichselbaum, Annals of Physics 327, 2972-3047 (2012).

- `QS`, $\mathrm{U(1)_{charge} \times SU(2)_{spin}}$, i.e., conservation of charge and full invariance in the spin space.
- `QSZ`, $\mathrm{U(1)_{charge} \times U(1)_{spin}}$, suitable for problems with external magnetic field and uniaxial magnetic anisotropy.
- `ISO` and `ISO2`, $\mathrm{SU(2)_{isospin} \times SU(2)_{spin}}$, the two differ in the phase convention in the isospin operator definition between the two leads in the two-channel case. Simplifying a bit, one should use `ISO` for problems where the first sites of the two Wilson chains belong to the same A/B sublattice, but `ISO2` for problems where they belong to different sublattices. This is appropriate for models at the particle-hole symmetric point (and bipartite lattice as regards the electron hopping terms) with full invariance in the spin space.
- `QSLR`, similar to `QS`, but only for left-right symmetric two-channel problems. There is an additional $\mathrm{Z_2}$ parity quantum number. Not that not only the Hamiltonian, but also all operators need to be reflection invariant. The spectral functions thus need to be computed from even and odd combinations of creation/annihilation operators.
- `QSZLR`, left-right symmetric two-channel variant of `QSZ`.
- `ISOLR` and `ISO2LR`, left-right symmetric two-channel variant of `ISO` and `ISO2`.
- `ISOSZ`, $\mathrm{SU(2)_{isospin} \times U(1)_{spin}}$, for problems at the particle-hole symmetric point and external field.
- `ISOSZLR`, left-right symmetric two-channel variant of `ISOSZLR`.

- `U1`, $\mathrm{U(1)_{charge}}$, for problems with no spin symmetry.
- `SU2`, $\mathrm{SU(2)_{isospin}}$, for problems with no spin symmetry at the particle-hole symmetric point.
- `SPSU2`, $\mathrm{SU(2)_{spin}}$, for problems with full SU(2) spin invariance, but no charge conservation (superconductivity within the BCS theory).
- `SPU1`, $\mathrm{U(1)_{spin}}$, for problems with U(1) spin symmetry and no charge conservation.
- `SPU1LR`, left-right symmetric two-channel variant of `SPU1`.
- `SL`, spinless fermions with conserved charge, $\mathrm{U(1)_{charge}}$.
- `SL3`, three-channel spinless fermions with conserved charge in each channel separately.
- `DBLISOSZ`, two-channel problems with isospin symmetry in each channel separately and U(1) spin symmetry, $\mathrm{SU(2)_{isospin,1}} \times \mathrm{SU(2)_{isospin,2}} \times \mathrm{U(1)_{spin}}$.
- `DBLSU2`, two-channel problems with isospin symmetry in each channel separately, $\mathrm{SU(2)_{isospin,1}} \times \mathrm{SU(2)_{isospin,2}}$, but no spin symmetry.
- `ANYJ`, $\mathrm{U(1)_{charge}} \times \mathrm{U(1)_{spin}}$, with arbitrary spin of the conduction-band electrons.
- `NONE`, no symmetry.

# Diagonalization

- Full diagonalizations with `dsyev/zheev`
- Partial diagonalizations with `dsyevr/zheevr`
  (possible when CFS/FDM is not used)
- For most problems this is where the largest amount of the processor time is spent.
- Note: symmetric eigenvalue problem has a high memory to arithmetic ratio. Unclear if GPUs would help much for large problems.

# Recalculation of operators

$$\langle QS\omega \| \hat{O} \| Q'S'\omega' \rangle_{N+1} = \frac{\langle QSS_z\omega | \hat{O}_\mu | Q'S'S'_z\omega' \rangle_{N+1}}{\langle S'S'_z; M\mu | SS_z \rangle}$$

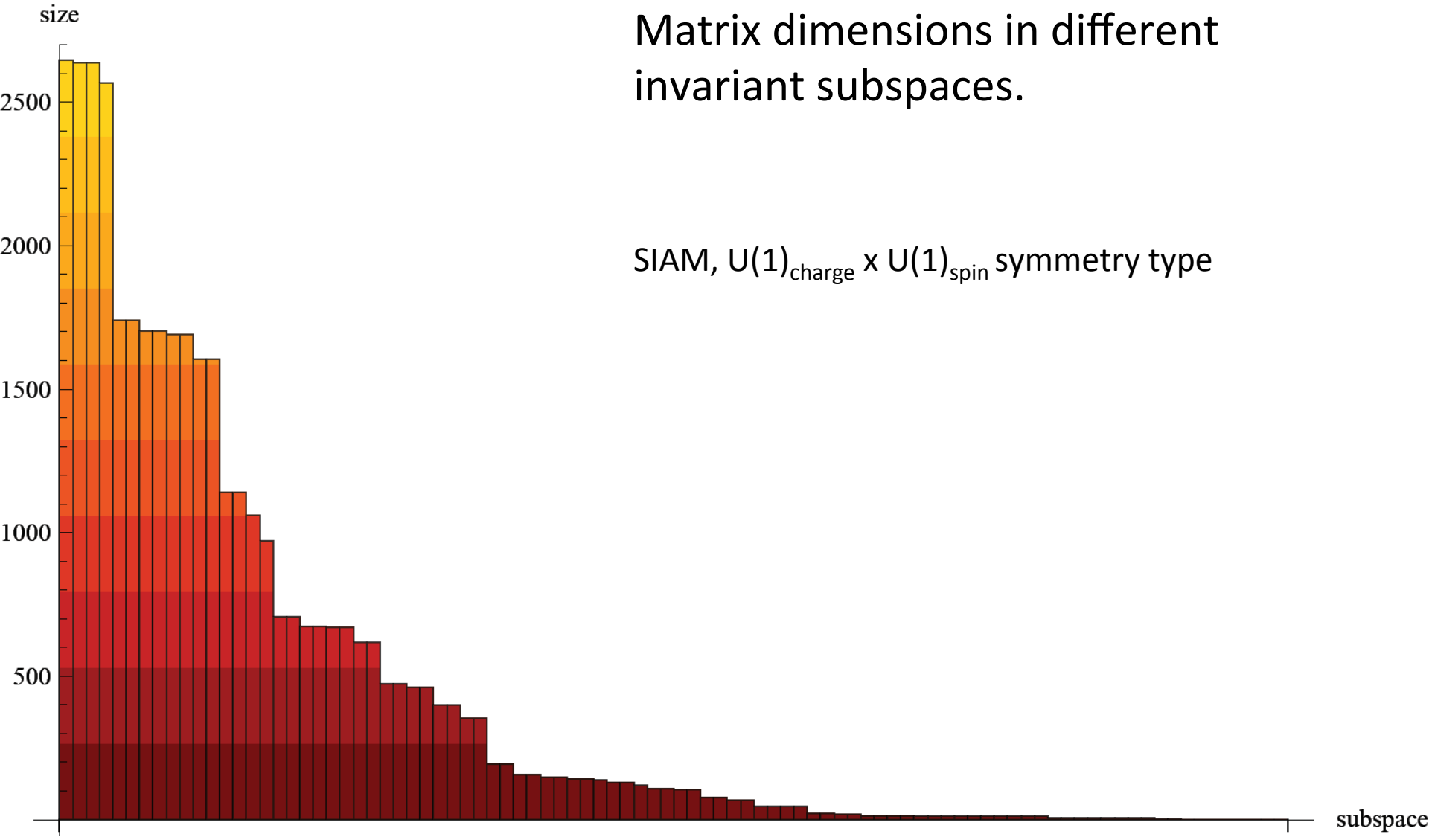
$$\langle QS\omega \| \hat{O} \| Q'S'\omega' \rangle_{N+1} = \sum_{ii'} C(QS, Q'S', ii') \sum_{rr'} U_{QS}(\omega, ri) U_{Q'S'}(\omega', r'i') \langle F_i(QS)r \| \hat{O} \| F_{i'}(Q'S')r' \rangle_N$$

Important to be efficiently implemented! We use BLAS routine **GEMM** (general matrix multiply). (GEMM from Intel MKL library has >80% efficiency on Xeon processors.)

| Level | Data Movement | Floating-Point Operations | Example |
|---|---|---|---|
| Level 1 BLAS | O(N) | O(N) | DDOT |
| Level 2 BLAS | $O(N^2)$ | $O(N^2)$ | DGEMV |
| Level 3 BLAS | $O(N^2)$ | $O(N^3)$ | DGEMM |

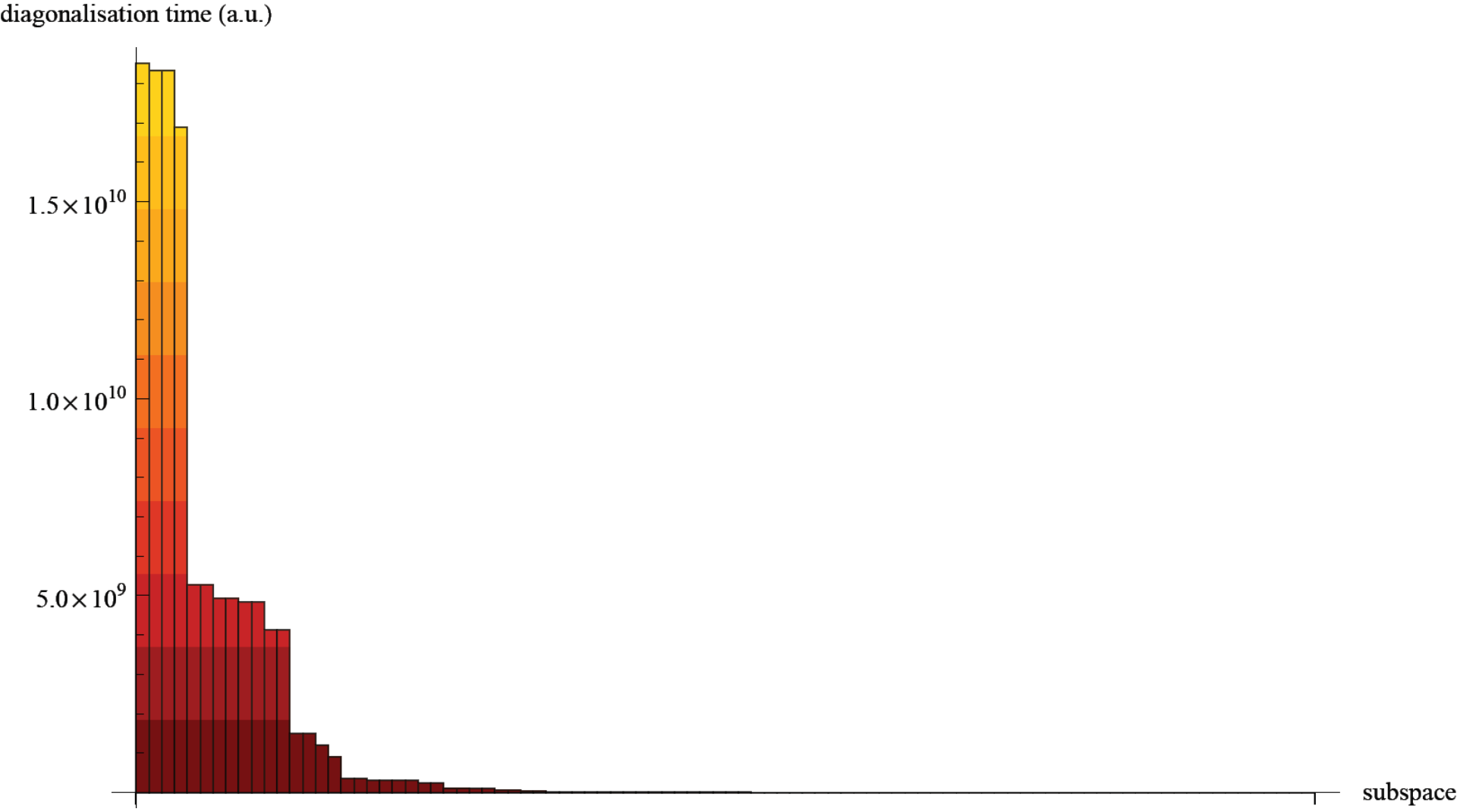
# Parallelization

- Multi-threading on multi-processor computers (pthreads or OpenMP).
  - Intel MKL implementation of LAPACK takes advantage of multi-core CPUs.
  - DSYEV does not scale linearly, but there is some speedup.
- Parallelization across multiple computers using message passing (MPI).
  - Parallel diagonalisations using LAPACK, or parallelized diagonalisation using ScaLAPACK.




# Matrix dimensions in different invariant subspaces.

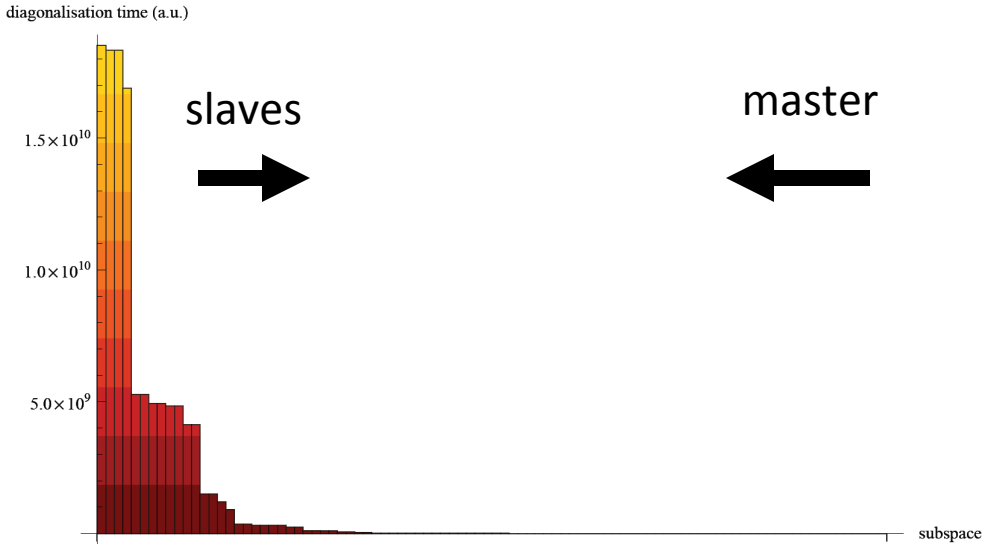
SIAM, $U(1)_{charge}$ x $U(1)_{spin}$ symmetry type




Conclusion: up to ~5-6 simultaneous diagonalisations.

# Master-slave strategy using MPI

- Master delegates diagonalisations of large matrices to slave nodes.
- Master diagonalizes small matrices locally.




Communication overhead is negligible!

# OpenMP + MPI

- Best so far: spread calculation across 5-6 nodes, use multi-threaded DSYEV on each node (4 threads).
- More recently: 4 CPUs, 8 threads each.
- TO DO: evaluate ScaLAPACK on machines with fast interconnect (such as Infiniband).

## **Nested parallelism** with OpenMP & Intel MKL

```
OMP_NESTED=TRUE
OMP_NUM_THREADS=4
MKL_NUM_THREADS=16
MKL_DYNAMIC=FALSE
```

Significant improvement, when it works!
(Segmentation faults,...)

# Toy implementation of NRG

- http://nrgljubljana.ijs.si/nrg.nb
- Implements SIAM in (Q,S) basis, it computes flow diagrams, thermodynamics and expectation values
- Reasonably fast (Mathematica internally uses Intel MKL libraries for numerical linear algebra and there is little overhead)

```
bz = qszbasisvc[{d[], f[]}]; MatrixForm[bz]
subspaces[0] = bz[[All, 1]];
showmatrices[makematricesbzvc[H, bz]]
Hrescaled = H faktor / Sqrt[Λ];
hams[0] = makematricesbzvc[Hrescaled, bz] //. params;
```

$$\begin{pmatrix} \{-2,\ 0\} & \{|\,\square\,\square\,\square\,\square\,\rangle\} \\ \left\{-1,\ -\frac{1}{2}\right\} & \{|\,\square\,\square\,\square\,\blacksquare\,\rangle,\ |\,\square\,\blacksquare\,\square\,\square\,\rangle\} \\ \left\{-1,\ \frac{1}{2}\right\} & \{|\,\square\,\square\,\blacksquare\,\square\,\rangle,\ |\,\blacksquare\,\square\,\square\,\square\,\rangle\} \\ \{0,\ -1\} & \{|\,\square\,\blacksquare\,\square\,\blacksquare\,\rangle\} \\ \{0,\ 0\} & \{|\,\square\,\square\,\blacksquare\,\blacksquare\,\rangle,\ |\,\square\,\blacksquare\,\blacksquare\,\square\,\rangle,\ |\,\blacksquare\,\square\,\square\,\blacksquare\,\rangle,\ |\,\blacksquare\,\blacksquare\,\square\,\square\,\rangle\} \\ \{0,\ 1\} & \{|\,\blacksquare\,\square\,\blacksquare\,\square\,\rangle\} \\ \left\{1,\ -\frac{1}{2}\right\} & \{|\,\square\,\blacksquare\,\blacksquare\,\blacksquare\,\rangle,\ |\,\blacksquare\,\blacksquare\,\square\,\blacksquare\,\rangle\} \\ \left\{1,\ \frac{1}{2}\right\} & \{|\,\blacksquare\,\square\,\blacksquare\,\blacksquare\,\rangle,\ |\,\blacksquare\,\blacksquare\,\blacksquare\,\square\,\rangle\} \\ \{2,\ 0\} & \{|\,\blacksquare\,\blacksquare\,\blacksquare\,\blacksquare\,\rangle\} \end{pmatrix}$$

$$\begin{pmatrix} \{-2,\ 0\} & \left(\frac{1}{2}\,(U-2\,\delta)\right) \\ \left\{-1,\ -\frac{1}{2}\right\} & \begin{pmatrix} \frac{1}{2}\,(U-2\,\delta) & \sqrt{\gamma} \\ \sqrt{\gamma} & 0 \end{pmatrix} \\ \left\{-1,\ \frac{1}{2}\right\} & \begin{pmatrix} \frac{1}{2}\,(U-2\,\delta) & \sqrt{\gamma} \\ \sqrt{\gamma} & 0 \end{pmatrix} \\ \{0,\ -1\} & (0) \\ \{0,\ 0\} & \begin{pmatrix} \frac{1}{2}\,(U-2\,\delta) & -\sqrt{\gamma} & \sqrt{\gamma} & 0 \\ -\sqrt{\gamma} & 0 & 0 & -\sqrt{\gamma} \\ \sqrt{\gamma} & 0 & 0 & \sqrt{\gamma} \\ 0 & -\sqrt{\gamma} & \sqrt{\gamma} & \frac{U}{2}+\delta \end{pmatrix} \\ \{0,\ 1\} & (0) \\ \left\{1,\ -\frac{1}{2}\right\} & \begin{pmatrix} 0 & -\sqrt{\gamma} \\ -\sqrt{\gamma} & \frac{U}{2}+\delta \end{pmatrix} \\ \left\{1,\ \frac{1}{2}\right\} & \begin{pmatrix} 0 & -\sqrt{\gamma} \\ -\sqrt{\gamma} & \frac{U}{2}+\delta \end{pmatrix} \\ \{2,\ 0\} & \left(\frac{U}{2}+\delta\right) \end{pmatrix}$$

```
nulmat[dim1_, dim2_] := Table[0, {dim1}, {dim2}];
mfsubsp[n_] := mfsubsp[n] = mf[n][[All, 1]];
fget[n_, subsp1_List, subsp2_List] := fget[n, subsp1, subsp2] = Module[{ndx},
    ndx = ElementPosition[mfsubsp[n], {subsp1, subsp2}];
    If[ndx =!= {}, mf[n][[ndx[[1, 1]], 2]],
     nulmat[dimsub[truncn[n], subsp1], dimsub[truncn[n], subsp2]] ]
   ];

prevn[n_] := truncn[n - 1];(* Subspaces in previous iteration after truncation! *)
oldsubspaces[n_] := subspaces[prevn[n]];
subspaces[n_] := subspaces[n] = Module[{oldsubs = oldsubspaces[n]},
    Union @ Flatten[Outer[Plus, oldsubs, qndiffs, 1], 1]
   ];
ancestors[n_, subsp_] := ancestors[n, subsp] = Module[{l},
    l = Map[subsp - # &, qndiffs];
    Map[If[MemberQ[oldsubspaces[n], #], #, Null] &, l]
   ];

ξ[n_] = (1 - Λ^(-n - 1)) / Sqrt[(1 - Λ^(-2 n - 1)) (1 - Λ^(-2 n - 3))];

makematrix[n_, subsp_] := Module[{anc, mat},
   anc = ancestors[n, subsp];
   anc = MapIndexed[{#1, First[#2]} &, anc];
   anc = Select[anc, First[#] =!= Null &];
   mat[i_, i_] := Sqrt[Λ] DiagonalMatrix @ e[prevn[n], anc[[i, 1]]];
   mat[i_, j_] /; i < j := mat[i, j] = ξ[n - 1] matrixhop[anc[[i, 2]], anc[[j, 2]]] fget[
   mat[i_, j_] /; i > j := Transpose[mat[j, i]];
   ArrayFlatten @ Table[mat[i, j], {i, Length[anc]}, {j, Length[anc]}]
  ];
hams[n_] := hams[n] = Map[{#, makematrix[n, #]} &, subspaces[n]];
```

```
spaghetti[nmax_, levels_, start_:1] := Reverse /@ Flatten[Table[Riffle[Take[energies[n], levels], n] ~Partition~ 2, {n, start, nmax, 2}], 1];

ListPlot[spaghetti[NMAX, 20, 1]]
ListPlot[spaghetti[NMAX, 20, 2]]
```